MASTER'S THESIS

Using metrics mined from group programming repositories to assist in individual guidance

Sandee, J (Jan Jaap)

Award date:
2020

USING METRICS MINED FROM GROUP PROGRAMMING REPOSITORIES TO ASSIST IN INDIVIDUAL GUIDANCE

by

Jan Jaap Sandee

in partial fulfillment of the requirements for the degree of

Master of Science
in Software Engineering

at the Open University, faculty of Science
Master Software Engineering
to be defended publicly on October 13th, 2020 at 11:00 AM

Student number:

Course code:	IM0502	
Thesis committee:	dr. ir. Fenia Aivaloglou (supervisor),	Open University
	dr. Bastiaan Heeren (chairman),	Open University

ACKNOWLEDGEMENTS

I want to acknowledge the unwavering support from Linda Hiemstra, who supported me throughout these years of study and helped me keep motivated. I would also like to acknowledge Paul Goolkate voor motivating me to pursue this master study. A big thank you to Gerralt Gottemaker for joining me on this study and great collaborations.

Of course my appreciation goes out to the teachers and students that assisted in this research, without whom this would not have been possible.

Lastly I would like to voice my appreciation for my supervisor Professor Fenia Aivaloglou, her motivation and critical eye helped me work on a research I can be proud of.

ABSTRACT

Group work projects are a common aspect of software engineering courses in higher education. In these group projects multiple students work on the same assignment to deliver a single software product. These projects serve as both a way of showing their ability, to continue learning their craft, and learning to work in a group. However, it is not a given that each student participates equally or learns new things in relation to programming.

An experimental evaluation was performed at Saxion Hogescholen HBO-ICT in the first year Software Engineering class. In this experiment a tool was introduced in a group programming course that mined metrics from Git repositories and displayed them to students.

The following metrics were analyzed and presented: Lines of code, comments and documentation (.md files) were split according to added, modified and deleted. Git Blame was used to display a division of last modified code by students. Introduction of method complexity and size were displayed, and lastly time of commit was displayed in a plot.

In the course students were required to develop a game in Android over the course of 3 sprints. Teachers were asked to use the tool, sharing it with students to help in their guidance. At the end of the project teachers were interviewed following a semi-structured interview and students were asked to fill out a survey.

The interviews were labeled according to themes and categories. The survey was analyzed and filtered. These results were structured according to three main topics, teachers, students and metrics.

The primary approach by teachers was discussing the metrics with students and asking them to explain the metrics that are being presented. They also used it as way of explaining concepts of software quality using examples from the report. There was a strong desire to use a system such as this in future projects as it saved time.

The overall response of students was positive. Students who saw the tool very late in the project stated they would have liked to have seen it sooner. General responses was being able to see who did how much was seen as positive.

Lines of code, complexity, and Method Size yielded the biggest response among both teachers are students. Lines of Code gave a reasonable indication of productivity, while complexity and method size introduced quality concepts to students. These metrics also allowed teachers to ask more directed questions during guidance sessions.

While a reasonable number of metrics were examined and tested, it was clear that lines of code due to its simplicity is easy to understand and process. Combined with that complexity and method size allowed teachers to compare the lines of code with the ‘quality’ of the code being delivered.

The Hawthorn effect creates a potential positive effect of students being more honest in their productivity, due to student knowledge of being monitored.

However, it was very clear from responses that method such as this should never be used directly in grading. It should always require a teacher interpreting and discussing the metrics with the students.

Recommendations for the future involve better research into which metrics are most suitable in this context. Additionally student behavior could be researched more in depth in regard to how it affects them and whether or not they should have access to the tool.

Contents

1

INTRODUCTION

Group work is not just delivering a product, it is also learning and improving skills. In software engineering the product is often a program too large for one person to make. This gives students an opportunity to further hone their skills and gain experience working on larger software projects.

Group work projects are a common aspect of software engineering courses in higher education. In these group projects multiple students work on the same assignment to deliver a single software product. These projects serve as both a way of showing their ability, to continue learning their craft, and learning to work in a group. However, it is not a given that each student participates equally or learns new things in relation to programming.

Related work shows that work is often not divided equally [14, 6, 3], while students do not always inform their teacher of this problem. Alternatively they simply divide the work such that they can get a passing grade. It has also been found that the role of the teacher is important to help students view group work in a positive light [3].

For a teacher to properly guide students in this they would have to analyze the code written by each team member, which is very time consuming. One advantage that software projects have, however, is that they are usually maintained using a version management system like Git. This allows tracking the code contributions students made over the course of the assignment.

In this thesis we will look at a potential solution to assist teachers in guiding students in their individual contributions during a project. This solution is in the form of a tool that runs static code analysis on the Git repositories of student projects. The resulting metrics can then be used by the teacher to identify imbalances in contributions and pinpoint which students are struggling with what.

The proposed tool was implemented and utilized during the 2020 run of the course 'Playground' at Saxion University of Applies Science. The course is the last project of the first year and was attended by 147 first year students and supervised by 7 teachers.

Firstly utilizing interviews with the teachers we will explore which metrics they benefited from the most in guiding their students, secondly we will run a survey among students to find out their experiences with these metrics being tracked.

The primary research question is formulated as follows:

In the context of group programming assignments in higher education, which metrics and visualizations are suitable and useful in assisting monitoring and guiding individuals in a group by teachers?

The following sub questions are used to answer the primary research question:

RQ1: How do teachers utilize these metrics within their guidance?

RQ2: How do students perceive the usage of tools monitoring contributions?

RQ3: How are utilized metrics understood by teachers and students?

By answering these questions the researcher intends to examine potential benefits of using a tool like this in group project. Rather than running simulations on public repositories, we can see how teachers and students respond to the usage of a tool like this. While also gaining insights into which metrics actually matter to the teachers and can help the students.

This thesis is structured as follows: In chapter 2 we explore the related work and background. In chapter 3 the methodology for the interviews and survey are explained. Chapter 4 we examine the development of the tool that was used in this experiment. This is followed by the results of the experiment in chapter 5. We then discuss these results in chapter 6, following by concluding words in chapter 7

2

Background and Related Work

The proposed research covers four specific domains within the context of Software Engineering: Group Work, Professional World, Use of Metrics in CS Education, and Metrics and Software Quality. In this chapter we will explore research that has been done into these domains related to the proposed research.

2.1. Group Work

Group work is a common method within higher education to teach students how to work with others and to apply the knowledge they have learned. However, group work also provides the opportunity to apply the content they know in a different setting. According to Johnson et al., two key ingredients in successful group work is interdependence and personal accountability [14]. Interdependence means that students need each other to receive a passing grade, while at the same time being held personally accountable for their actions.

The common practice withing group projects is to assign every student the same grade. With every student receiving the same grade this can result in unwanted behavior from students while they work on the project. Examples are students who do not do enough work, usually referred to as slackers [14, 6, 3]. Alternatively, students might decide to do all the work themselves as they do not want to risk a failing grade. In a qualitative analysis using interviews with 65 students, Colbeck et al. found that students are hesitant to rat out their fellow students, with avoidance being the main strategy they employ [6]. They instead found that most students in groups that received the same grade resulted in one or two people doing all the work. There was a notable difference in their goals within a group based on prior experiences, where some people only wanted to get a good grade regardless of learning experiences.

It was found that the instructor plays an important role in guiding students and making them see the learning effect. Utilizing surveys among students administered by instructors, Chapman and van Auken found that as the instructor's role is seen as positive, concerns for work and grade equity decreases [3]. They also found that when dividing students into three groups, the medium and high scoring groups were all positive about the group experience, while the low scoring students were negative about the experience, but also about the instructor.

One of the most active research topics related to group work is properly measuring individual contributions. Various other methods have been proposed that focus more on how

the group performs utilizing analysis of logs and running natural language processing on reports [13, 16, 4, 20]. In all cases the focus was on managing written logs and utilizing other techniques for assessing participation. Other methods include utilizing a dedicated social platform, tracking all the issues, holding peer reviews, or even examining the code. These are all tools that are available when trying to grade students and holding them individually accountable.

2.2. Professional World

It should be noted that in the professional world measurement is not focused on seeing equal contribution. This section was included to examine which metrics are common and what their effects are. These are summarized in Table 2.

Within software engineering it is standard practice to maintain a code base using a version control system such as Git. This creates a history of each change made to a program and by whom it was made. In the business world this has been used to measure productivity of employees. Lima et al. looked at metrics of 48 developers over a period of twelve weeks, looking at code contribution, average complexity, bugs introduced and bugs fixed, then presented the results to the managers of each team in an interview [17]. The results were then processed using the grounded theory [7]. Of those metrics, code contribution and average complexity were seen as useful by project managers, while bug introduction and repair were received more skeptically. Complex jobs tended to introduce more bugs and as such a developer that introduced more bugs was more likely working on a complex job rather than having trouble with programming in general.

Within the business world it is also understood that working on a project does not just involve writing code. Employees also spend a lot of time performing what are known as clerical activities, such as writing documentation, checking tasks in planning software, writing bug reports, etc. [12]. A model for measuring contribution was made based on these factors (see Table 1 for code related factors), creating a more broad measurement. This was only proposed as a plugin, but implementation has not been found.

Interestingly job satisfaction is related more closely to the role of the manager. Utilizing a survey among 465 software developers, Storey et al. found that actual code quality does not seem to have a strong relation job satisfaction [27]. However, it was found that personal perceived productivity could vary wildly from actual productivity.

Introducing metrics to track productivity is not a trivial task as the introduction of such metrics can quickly be perceived as goals to meet, rather than indicators of something requiring attention [28]. Developers will instead focus their entire work on reaching those targets without actually thinking about writing quality software. However, when people are being observed they will change their behavior, even if there is no right or wrong, this is known as the Hawthorne effect.[1]

2.3. Use of Software Metrics in Computer Science Education

Research has been done into using static code analysis metrics to assist students in learning to program. Utilizing a set of metrics (see Table 2) Cardell-Oliver created custom messages

[1] Wikipedia: Hawthorne Effect, http://en.wikipedia.org/wiki/Hawthorne_effect

for unit tests, Checkstyle and PMD, software designed to scan for common coding and style errors, to give formative feedback to students making assignments [2]. This resulted in being able to identify which students are non-starters, stoppers, movers and tinkerers [24]. They introduced a set of suggestions for programming courses such as: Clear criteria from instructors, students should be encouraged to refactor and improve code, and refactoring and quality improvements should tie into professional work. Blok and Fehnker applied a similar method of creating custom messages for PMD to assist students in learning to program [1]. The work was only tested on open source software as a proof of concept of how many warning would be returned specifically for a novice developer.

McGill and Patton suggest using portfolios with slowly improving numbers and metrics can be beneficial for a student to track their growth [23]. They also suggest expanding well-known metrics with pedagogical metrics: Language Vocabulary, Orthagonality/Encapsulation, Decomposition/Modularization, Indirection and Abstraction, Polymorphism, however, they do not specify how to measure these metrics.

Letting students see their improvement based on metrics can be considered a form of gamification. Dubois and Tamburrelli found, in a small preliminary experiment among 2 groups of 32 students, that this helped improve achieving the metrics. [9]. One group saw only their own metrics and focused on maximizing metrics favored by teaching assistants, while the other group could also see the metrics of other groups. This resulted for instance in reduction of lines of code as they saw other groups achieving similar goals with less lines.

Creating a suite of metrics for mining Git repositories in education has been done, focusing mainly on volume of code being added per month. Parizi et al. created a set of metrics for mining Git repositories to measure participation [22]. These metrics were tested as a proof of concept on a large open source project indicating differing contributions from developers. They also tested an algorithm to estimate time spent per day on coding based on Git commits.[2]

2.4. Metrics and Software Quality

When it comes to metrics, many different collections and sets have been created. Many focus on procedural programming, but specific metrics for object-oriented programming have also been developed [5]. However, the two metrics that seem to show up with most frequency are Lines of Code.[3] and Cyclomatic Complexity [19]

Lines of Code have been shown to be language agnostic and easy to calculate [21]. It is important to filter out duplicate code and boilerplate code, but when doing so gives an indication of how much code someone contributed. It is also an important factor in recognizing problematic code artifacts such as methods that are too long. By analyzing 26 semi-automatically chosen projects against a large suite of well known metric tools, Gil and Lalouce found that metrics are often coupled to size to be any kind of predictor in regards to quality [11].

Many tools have been made to calculate software metrics, though it has been observed that each tool can have different names and interpretation for the same metrics. Lincke et al. tested 10 different metric analyzing tools utilizing 9 object oriented programming based metrics (distilled from a list of apparently 200 available metrics) on three large open source

[2] Git Hours, https://github.com/kimmobrunfeldt/git-hours
[3] Wikipedia: Source Lines of Code, https://en.wikipedia.org/wiki/Source_lines_of_code

Action	Type
Add lines of code of good/bad quality	+/-
Commit new source file or directory	+
Commit code that generates/closes a bug	+/-
Add/Change code documentation	+
Commit fixes to code style	+
Commit more than X files in a single commit	-
Commit documentation files	+
Commit translation files	+
Commit binary files	-
Commit with empty commit comment	-
Commit comment that awards a pointy hat	+
Commit comment that includes a bug report number	+

Table 1: Actions that can be performed on code and whether it is positive (+) or negative (-)[12]

projects [18]. The results suggested that interpretations of metrics in tools and results could vary wildly. Similarly, Fernandes et al. found that tools designed to locate code smells use different measurements [10]. They examined 84 tools listed in 107 studies against the same open source project. Ultimately, they only tested for Long Method and Large class with 4 tools (from the original 84) that applied the same rules. This resulted in finding differences in those tools.

Gousios et al. [12] proposed a model to be implemented on a large platform	• Lines of Code • Contribution Factor Function (see Table 1 for a list of code related factors)
Lima et al. [17] looked at SVN libraries over a period of 12 weeks	• Code Contribution • Average complexity per method • Introduced bugs • Bug fixing contribution
Cardell-Oliver [2] looked at Automated Feedback generated from student exercises.	• Program Size (effectively LoC) • Functional Correctness • Efficiency (execution speed) • Program Style (With PMD[4] and CheckStyle[5]) • Client Validation
Dubois and Tamburrelli [9] used SonarQube metrics on an Android based boardgame assignment.	• Lines of Code • Percentage of sonar rules compliance • Percentage of branch coverage of test cases • Percentage of JavaDoc documentation
Parizi et al. [22] tested a suite of Git mining related metrics on an open source project	• Number of commits per month • Number of merges per month • Number of files per month • Total lines of code per month • Time spent on the project per day
Lincke et al. [18] tested these metrics using 10 different tools on 3 large open source software projects. Each metric is on a per class basis.	• Coupling between object • Depth of inheritance tree • Lack of cohesion of methods • Lines of Code • Number Of Children • Number Of Methods • Response For a Class • Weighted Methods per Class (CC as method weight)

Table 2: Cited papers and the metrics they used, names of metrics taken directly from the papers

3

METHODOLOGY

From the related work we have learned that the instructor (and even manager) plays a vital role in a group project in regards to the satisfaction and learning objectives of the student. However, analyzing code manually is time-consuming for an instructor and even then they might miss important details.

Within software development we have a unique opportunity to assist an instructor. When coding, students often utilize Git to collaborate and make changes to the software. This means that all changes and modifications made by students during a project are tracked. Utilizing repository mining and well-known standard metrics it should be possible to create an overview associated with each participant in a project. The goal is to create a tool that will automatically mine a repository that it is given and produce a, potentially interactive, report that will assist a teacher in guiding a project group.

This report would give a teacher insight into how different students are contributing to a project. Additionally the teacher can point out potential code quality issues if such metrics are being tracked. Most notably as has been pointed out regarding the use of metrics, it helps a teacher locate anomalies in the code.

A tool was developed to mine Git repos of school projects and generate a visualized reports for the teachers. Metrics were chosen based on their known accuracy from existing research and their availability in existing software. This tool was developed incrementally, focusing primarily on Lines of Code, but also examining complexity. An initial version was presented given small modifications based on feedback from the teachers. Halfway through the project significant changes were made based on usage feedback from the teachers. This last version was the primary focus of the experiment. A more exact description of the tool can be seen in chapter 4.

3.1. EXPERIMENTAL EVALUATION

To test this tool an experimental evaluation was done during a school project. The tool was introduced into a course at Saxion Hogeschool where first year students have to develop a simple mobile game in Android. The course ran for seven weeks, from May 4th until June 26th 2020. In the first class students each pitched a game idea. Of these ideas the most popular were chosen and groups were formed. Then students had 3 sprints to develop the game. Students were required to work using a Git repository hosted on Gitlab provided by the institution, and this repository was available to the researcher and 6 other teachers.

There were a total of 7 classes spread out over 2 locations (Enschede and Deventer), with 3 classes being part of the English taught international track. Each class consists of about 24 students. There were in the end a total of 39 groups, with most groups consisting of 4 members. There were 2 groups of 2, 7 groups of 3, 28 groups of 4, and 2 groups of 5, giving a total of 147 students.

Guidance sessions were scheduled once a week by a teacher with about 4 hours total to guide up to 6 groups. There were a total of 7 teachers, the researcher included. Of these teachers only one was guiding 3 different classes, every other teacher only had one class. The teachers were given simple instructions on the working of the tool. How they then incorporated this into their guidance sessions was up to them. The reasoning for this was partly to observe how teachers would approach usage of such tooling, partly to allow teachers freedom in how they organized their guidance sessions.

Students were made aware of the tool being used through an information page on their course page. They were not given default access, instead they could contact the researcher to gain access. 7 groups made use of this option, though 5 of those in the last few weeks. Actual statistics of what and how much people looked at it were not tracked. However, almost all groups looked at the tool together with their guiding teacher. So the majority did in the end see the tool and the metrics.

The tool only utilized data mined git repositories that were assigned to them, teachers always have access to these repositories. The tool only displayed information from these repositories in a different way. Since this concerned internal data part of the education it could not be used directly for analysis in the research. However, 6 groups filled out permission forms to allow their pseudonymized data to be used within this research. These was mostly used for visual examples as the data set was not large enough to derive empirical data from it.

3.2. Data Collection and Analysis

The primary research design was a convergent parallel mixed method [8], a combination of quantitative data and qualitative data was used to answer the questions.

For *Qualitative Data Collection* structured interviews with the teachers were used. Which we look at first in Interview Data Collection

Quantitative Data Collection was done through a survey done among students at the end of the course. Which is examined next in Quantitative Data Collection

Interview Data Collection

The primary goal of the research is discovering how metrics can be used by teachers during their guidance. To this end all six teacher who taught the project were interviewed. The following will detail the interview and data analysis process.

Interview Process

Interviews were semi-structured. All participating teachers were asked for consent. The average length of the interviews was 40 minutes. The interview protocol can be seen in Appendix B. All interviews were performed using Microsoft Teams with a local setup for recording the entire interview.

Six teachers were interviewed. Of these two had only begun a few months before this class, two had 5 years of experience guiding projects, and the last two had 8 or more years of

experiences guiding projects. Of the experienced teachers only two had experience guiding projects in the first two years of the study. The others all had experience teaching fourth year students as well[6].

The semi-structured protocol features four major themes:

- *Project Guidance*, focusing on how they did project guidance beforehand.
- *Usage*, how and when did they implement the tool in their guidance.
- *Metrics*, their views on the metrics and how they were used.
- *Evaluation*, comments and thoughts on the usage and future of metrics project guidance.

The interviews were transcribed by hand by the researcher, focusing on transcribing for readability rather than exact accuracy. As such 'uh' and similar were not transcribed.

Data Processing

Each interview was labeled according to thematic analysis. The four themes from the interview protocol laid a good foundation and a fifth, Students, was added. This was due to teachers often noting student behavior and reactions during the interviews.

Labels were assigned on what was spoken. After the first two interviews no new labels were introduced in the coding process. A focus was placed on data relevant to the study.

For summarizing an intermediate step was used taking each label and summarizing what each teacher said on the matter. Finally from these labels summaries were made for each theme.

Some labels were not included here as they were not relevant for the research. This includes differences with business projects some teachers had experience in.

Eventually the themes were mapped to the research questions.

- RQ1: Project Guidance, Usage, and parts of Evaluation
- RQ2: Students
- RQ3: Metrics

Evaluation also resulted in responses that were interesting but did not directly answer the research questions. These were

Survey Data Collection

While the focus of the study is on the guidance of teachers using metrics, we also want to see what effect, if any, this monitoring method has on students. To this end, a survey was created to be filled out by students near the end of the course.

[6] In the HBO-ICT course at Saxion University of Applied Sciences the third year is reserved for internship and minor, while the fourth year features specialization projects

Survey Design

The survey was anonymous with each question presented in English and Dutch. A total of eight questions were created with the aim of making the survey quick and easy to complete. Appendix C shows all the questions as they were created. There are 6 quantitative questions and 2 open-ended questions allowing students room to voice their opinion. None of the questions were mandatory. The multiple choice questions were designed such that only one answer made sense. Demographic questions were omitted on purpose to maintain a focus on the primary questions.

The survey was designed following guidelines about the survey introduction, question phrasing and design and question ordering [8, 15, 25]. Questions were written in English and Dutch to accommodate both international and national students. We also start with a specific question regarding the frequency of use of the tool by the teacher (RQ1). Thematically following that we look at whether students utilized this information on their own (RQ2). Then questions about the metrics as presented and whether or not they were clear and reflected their work (RQ3). Given that pair programming might have affected their opinion on the metrics, it was important to find out if students did this frequently. Lastly we end with two open questions.

The survey was tested with three colleagues and four students. Their answers were subsequently removed and their feedback was incorporated into the survey.

Data Collection

The survey was made using Qualtrics, a survey tool available to Saxion Hogescholen. The survey was made public in the last week of the course. Teachers were asked to take a moment in their lessons to have students fill out the survey. In some cases students requested access to the tool which was then granted. At the end of the week a notice at the course page was made asking those who had not filled it out yet, to fill it out (this automatically sends an email). It was clearly communicated how much time (around 5 minutes) it took to fill out the survey. The total responses were 93 complete responses in which all questions were filled out. This means there was a response rate of 63% based on the total possible students that took the course. It should be noted that students guided by the researcher also belong to this group.

Data Filtering and Analysis

In the surveys, two students claimed that the teacher never showed them the tool or the report. Their responses could not have been informed by the use of the tool. Their results have been removed from the further results, leaving the final number of responses to 91.

The other quantitative questions were split according to frequency of having seen the tool, to see if there were major differences between students that saw it often versus students that only saw it a few times.

The open questions were labeled using qualitative data process methods.

4

THE TOOL

4.1. DEVELOPMENT

For the development focus was put on off the shelf tooling keeping in mind researcher experience with libraries and programming languages. There was a fairly short time frame in which the tool had to be developed to be ready for the experiment.

There was focus on developing the tool with iterative increments, allowing for feedback from the teaching team. Keeping in mind what was needed at a minimum, reports showing metrics mined from Git repositories, new features were added as time permitted. The main focus was having reports available for teachers to use in their guidance moments.

Once a basic version was done it was presented to the teachers involved, giving them an idea of what it looks like. This introduced two specific requests from the teachers:

- The ability to see what code belongs to whom in the final product
- What times are they committing.

These features were easily added due to the architecture setup of the tool, which was designed to be modular and easily modifiable. This resulted in the first version that was available during the execution of the course.

During the first few weeks of the tool being used various small alterations were made to improve accuracy of the data and visual updates. These included:

- In the initial version the tool only cloned the master branch and examined that. However, if students had been working on another branch that had not yet been merged into master it would appear as though they had not done anything. This was fixed by checking out all known branches. It should be noted that if a student never pushes that branch to the server it will never be known by the tool. These activities not showing up was validated by examining the event log on Gitlab itself.
- In some cases students had enabled a feature that created files not listed in the gitignore. This caused heavily inflated metrics as these files were constantly being modified. It was found to be one specific code file, which was manually ignored in the code to correct statistics.

During initial talks with students both the researcher and another teacher found themselves constantly clicking between added, changed, deleted to see productivity. A change was made in the visualization to place these bars side by side so that activity could be more accurately measured.

An attempt was made at displaying Added/Modified/Deleted in one view in the timeline, this resulted in a cluttered view. This view was only used for a few days before it was removed.

Halfway through the course there was a meeting among teachers to discuss progress of the course. During this meeting the state of the tool was also discussed. Four of the six teachers were present for this meeting. Notes were taken for each feedback point. Based on these notes serious changes were made to the presentation of the metrics. This resulted in effectively a new version of the tool, referred to as Version Two. Section 4.4 details what choices were made and why. Choices were not formed by consensus as each teacher had varying requests. Instead all the feedback was taken and examined with available options.

4.2. ARCHITECTURE

While designing the tool, one of the biggest issues was computation time. Presenting data does not really require a heavy computer. However, running static code analysis can quickly become very CPU intensive. As such the architecture was designed to be split up into three separate components. Figure 1 depicts the basic layout of the architecture, designed in such a way that the analyzer can run on a separate machine.

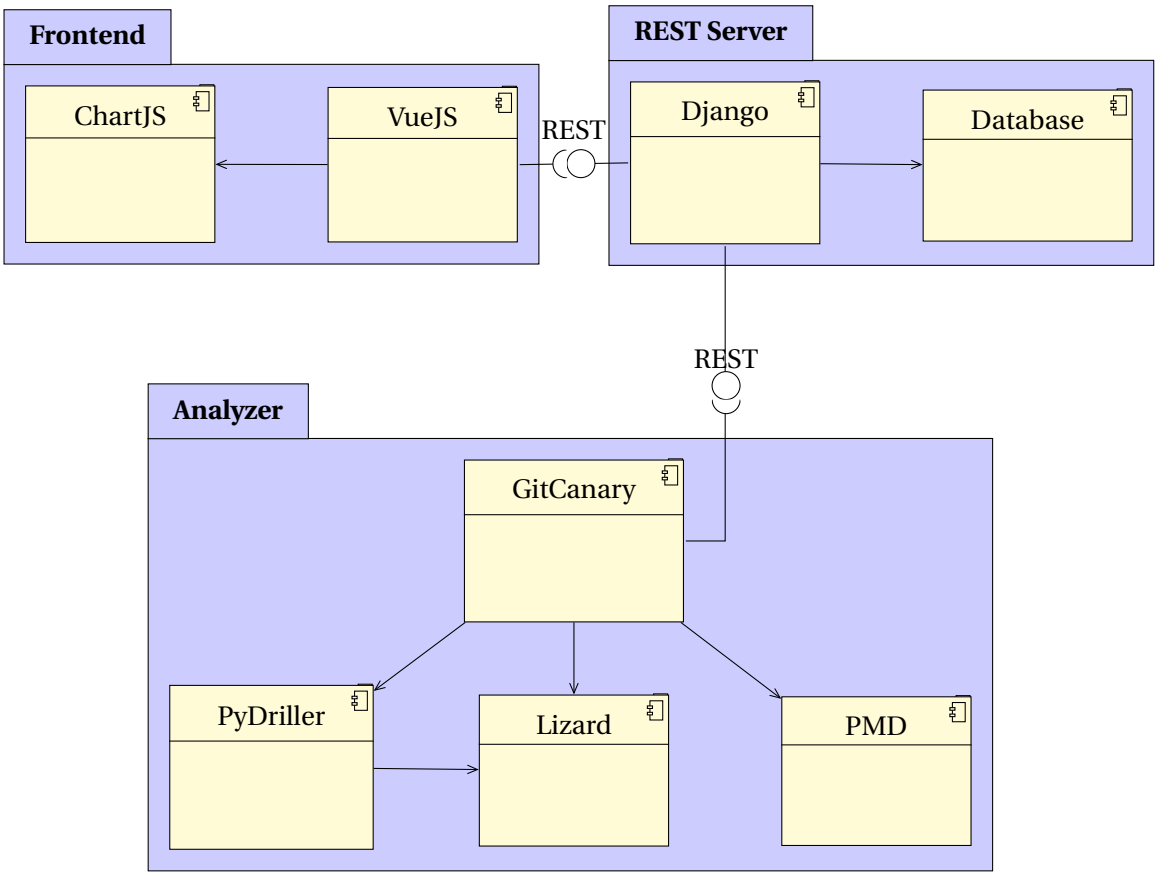

Figure 1: Tool Architecture, arrows indicate one direction usage

Frontend

For the frontend VueJS[7] was chosen to read information from the REST-server. This combined with ChartJS[8] allowed for visualizing the data as it was pulled from the server, using vue-chartjs[9] to speed up utilizing this.

Both versions of the frontend use the same libraries and basic structure.

Rest Server

Django (with Django Restframework) were chosen due to the researcher's familiarity with this system in getting a quick system running. Most notably things like login and users are already handled easily. The rest-server has a very basic data-structure for the actual reports. Figure 2 shows the database model that was used.

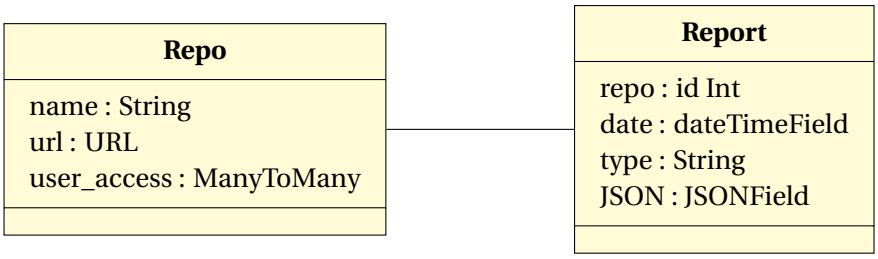

Figure 2: REST Data Structure

Analyzer

The analyzer examines the Git repo, generates a JSON report and submits it to the REST server. To know which repositories to examine it requests a list from the REST server and runs through it in order. It then clones the repo locally and makes sure all known branches are also available locally.

PyDriller [26] is the primary source of metrics which uses GitPython and Lizard to examine repositories. Lizard is used to obtain information such as Cyclomatic Complexity. It was configured to only examine .java or .md files while scanning since the researched project had all projects in Android Java.

While the tool can just loop through each commit via PyDriller, PMD[10] and CPD[11] require the actual code to be examined. For this the tool does a checkout on each commit and then PMD and CPD are run to examine the code.

[7] https://vuejs.org/
[8] https://www.chartjs.org/
[9] https://vue-chartjs.org/
[10] a static code analyzer -https://pmd.github.io/
[11] code duplication analyzer part of the PMD package

4.3. Frontend Version One

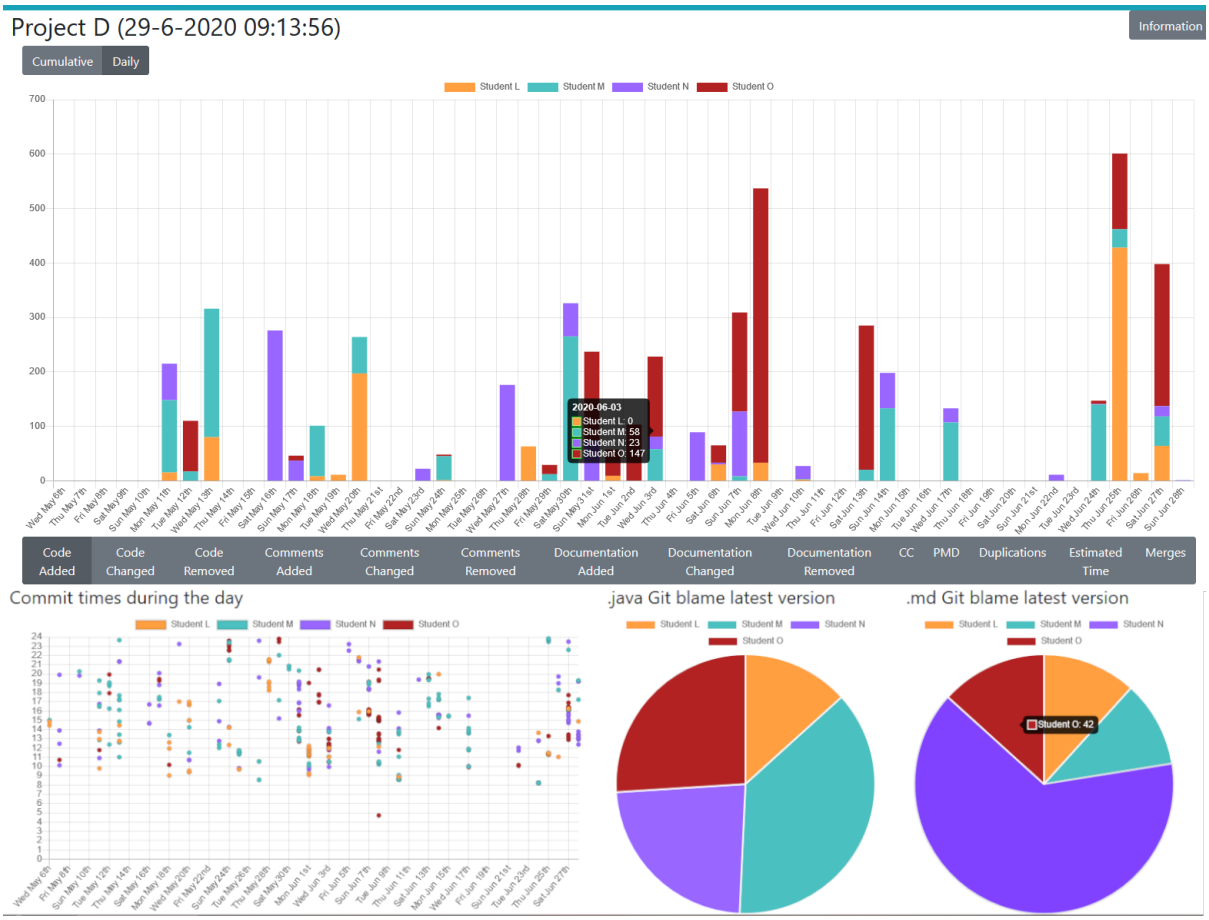

Figure 3: Front-end first version

The first version of the tool utilized a time line view as can been seen in Figure 3.

Chosen Metrics

The following factors were taken into account when looking at metrics to use.

1. What metrics were found in related work in Table 2.
2. What metrics can easily be extracted using existing software.
3. What metrics is the researcher interested in as a teacher himself.

Since the experiment revolves around testing the tool in an actual environment and potentially adjusting based on that, metrics were chosen based on experience as a teacher. Most of the related work did not consider placing metrics on a time line.

A focus was put on a day to day basis rather than a per commit basis, then divided per committer in the project. Table 3 shows the metrics used in the first frontend version and how they were calculated.

These metrics were all shown in a stacked bar graph per day, where the viewer can select which metrics they wish to see. Stacked was chosen to also display the total group effort

Metric	Calculation
Lines of Code	The amount of code introduced, changed or removed. This was achieved by matching added lines in a git commit to removed lines. If they were very similar then it was most likely a modification. Otherwise a new line or a removed line.
Comments/JavaDoc	The amount of comments/JavaDoc added, changed or removed. The method was similar to code. However, it was focused entirely on Java marking only lines starting with {/*, *, //}. While not entirely according to the Java specification of code commenting, convention dictates using only JavaDoc or single line comments, which is also what most students are used to.
Complexity	How much did the complexity change after this commit. By measuring the complexity of the previous code versus the new code how much did complexity change. The primary focus of this is seeing if people are writing code with any degree of complexity.
Quality warnings	Using PMD, how much did the number of warnings change from commit to commit. Meant to see who is introducing them and potentially who is fixing them. Students were given information on the utilized rules and how to examine these themselves.
Duplicate Code	Using CPD how many instances of code duplication of minimum 100 tokens were detected. If a piece of code was duplicated more than once this simply counted as one.
Non-code	How many non-code files were modified by a person. Regardless of how much it changed, simply modifying a file that was not code adds one. This was done to see if students are actually working on the project but not on code segments.
Markdown	Students were required to write all relevant documentation using markdown. The method of analyzing was similar to that of code and comments.
Merges	Merges were counted to see who was doing these and how frequent.
Estimated Time Spent	Utilizing an algorithm[12] get a sense of how much time is spent on each coding session.
Files touched	Which files was this developer involved in modifying. Per day it is possible to see which file a student worked on without any other information.
Commit Times	Using a scatter plot, each time of day of a commit was shown.
Git Blame	Using only the latest version of the master branch, Git Blame was used to display the number of lines of code and comments from each students who were the last to touch those lines of code.

Table 3: Metrics used in Frontend Version One

of that day as well as individual contributions. Additionally it is possible to switch between per day or cumulative so that it is possible to view total contribution over the entire project.

Commit Times were displayed in a separate graph using a scatter plot. Each dot shows a time someone committed. This was requested by a teacher on the grounds of being able to see if they are committing roughly at the same time and thus working as a group.

Git Blame was used to state the division of code in the most recent product. This was requested by a teacher on the grounds of being able to see if someone is producing a lot of code that either, never makes it into the final product, or is constantly rewritten by others. The same is done for the markdown files.

4.4. Frontend Version Two

Figure 4: Front-end second version overview

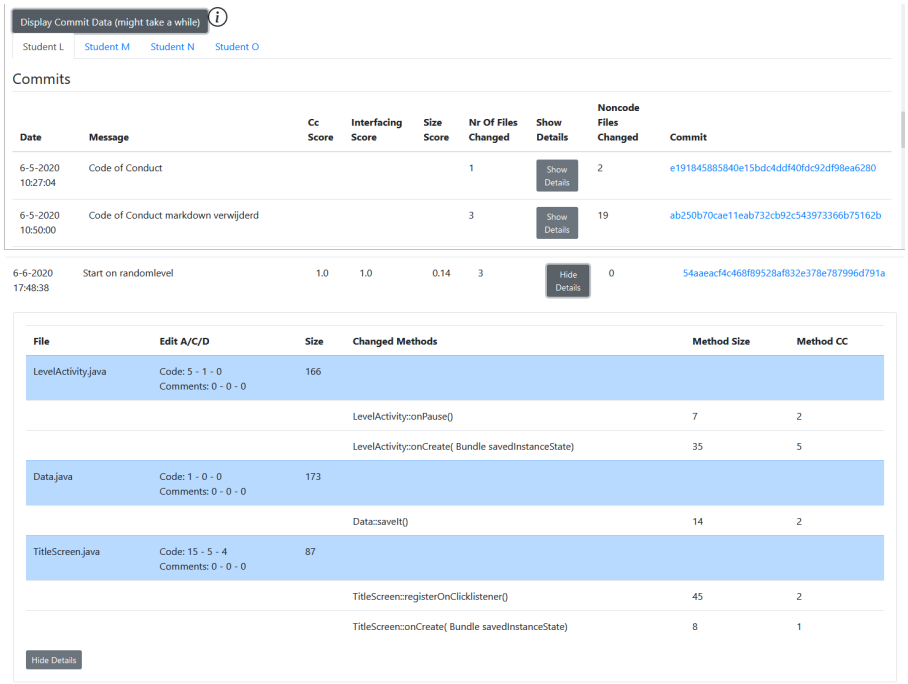

Figure 5: Front-end second version collapsible commit details

For the second version some drastic changes were made based on notes taken at a meeting with teachers. Firstly the layout was changed to not require clicking between graphs. Secondly the time line was dropped in favor of being able to select a date range. This resulted in a far clearer overview of all the metrics. Figure 4 shows everything as one continuous page, which was how it was displayed to teachers.

Some metrics were dropped for varying reasons, primarily based on feedback from the teacher looking at what metrics they focused on more during the meeting. Aside from stated reasons the metrics listed below also did not get any attention or interest from the teachers. For examples of how these metrics were represented refer to Appendix A.

1. *PMD* was removed as students did not have the time to properly work on this.

2. *Code Duplication* was removed as the projects were too small to introduce serious duplication problems. Also this and PMD were difficult to calculate and not having it reduced run time significantly.

3. *Estimated Time Spent* turned out to be woefully inaccurate. There were many instances of students only doing one commit on a day. The used algorithm relied on relatively frequent commits. Secondly teachers stated in the meeting they were not that interested in seeing this metric.

4. *Non Code Files* was not entirely dropped but moved to a position of less focus. Given that the aim to writing more code, this seemed unneeded.

5. *Merges* similarly moved as it was less interesting.

6. *Files Touched* was moved to a subsection of the commit list (Figure 5). As the original working was impractical.

Additionally Complexity was altered to not reflect a change in complexity for that day, as that proved to be too confusing, and was instead replace with the adding and altering of methods and their complexity.

The same was done for method size, allowing the option to see in what range methods were being added or left after editing.

The reasoning for both of these was being able to see if students were adding significant methods to the code. If a student only adds really small and simple methods or if a student only adds complex/large methods, this might be an indication for the teacher that an intervention is needed.

At the bottom of the screen there was a button which when clicked resulted in the display seen in Figure 5. The CC Score, Size Score, and Interfacing scores were delta maintainability numbers generated by PyDriller[13].

The version that was used during this experiment is available in open source on Gitlab[14]. Since this research it has been updated and improved into a newer version.

[13] https://pydriller.readthedocs.io/en/latest/deltamaintainability.html
[14] https://gitlab.com/jjsandee/gitreporter

5

RESULTS

In this section we will look at each of the sub research questions and map the results from the interviews and survey to the appropriate sections. For the interviews, the teachers have been randomly assigned labels T1 to T6. The answers have been translated from Dutch to English.

We first look at how teachers used the tool and what impact they felt it had, compared with how students reported usage in guidance sessions. Next we look at how students reacted to the tool looking at both teacher observations and student responses. We will examine the metrics that were used in the tool and how teachers used and saw them, and also how students found their clarity and accuracy. Finally we will look at how both students and teacher view future uses of this tool.

5.1. RQ1: TEACHER USAGE

How do teachers utilize these metrics within their guidance?

In order to understand how teachers used this tool, questions were asked regarding how they approached projects before. This gives an idea of how they view group work and how they differentiated students in the past. Then usage is examined based on interview and survey results. Lastly evaluation and potential future use were discussed with the teachers from their point of view.

BEFORE

All teachers with previous experience noted that in the early years, and especially early projects like this, students have little experience with group work. As such they feel students need a lot of guidance in terms of project management. It was specifically noted that students have little experience in communicating with each other (T2, T3). In later years this is less of a problem according to one teacher which allow for more focus on the technological side.

One teacher (T6) did note that in many projects there is not a lot of time available to properly dive into code to really discuss which software solutions are the best approach:

"I think that you want to do something with the content, with the code so to speak. ... So what you actually want to do is take the time to dig through the code. Spend a good hour to dig through everything. 'hey how did you do this? why did you do it like this? what is better, etc. That is missing, but you usually don't have the time for that."

All teachers tended to follow a fairly similar approach in project guidance. Some had a central start, but all essentially went to each group to witness their stand-up[15] meeting. Some teachers would first witness all the stand-ups consecutively and then make another round for additional comments, others just did each group in order doing stand-up and comments in one session but the overall described process of each teacher was the same. Witness stand-up, ask questions based on things heard during stand-ups, look at and discuss documents or parts of the program.

There were differences in how some of the teachers approach supervision and what their aims were. Some teachers (T1, T2) put heavy emphasis on project management and process in these sessions. One teacher noted wanting to get consensus so that students understood why things had to be in Git (T4). Two teachers (T4, T5) also noted they would take concepts that come up and turn them into a learning experience for the students.

It was again noted here by one teacher (T6) that the code often did not get covered extensively due to time constraints.

One teacher (T1) approached this by constantly asking questions about student activity in a group, which he noted was a very time consuming process. When asked if this was accurate he answered the following:

"Yes, for me it does. Because I could just keep asking questions. For me it was accurate. They could not get out of it. If I thought something was not right, I would keep asking. They did need to give more information."

Another teacher (T2) used a combination of noting what students said that they had done during a sprint, combined with information from other teachers about students to look out for. There was one teacher (T5) who on a previous project mostly looked at commits to get a feel of how much work each student did. Lastly a teacher (T6) noted that during stand-ups the student who did not do a lot would often be very quiet. This teacher also taught most of the programming courses and as such was generally aware of the grades that students possessed in previous classes.

There was one teacher (T3) who stated that this was a flaw in our course program. According to him, in later years in their studies it is less of a problem due to students being more professional about it. However, in early years freeloaders are an issue. He also asserts that, if a group functions properly, the end product cannot be broken up into separate parts, making it nearly impossible if the work was divided evenly:

"And I would say I always get it accurately, but usually I have to say if you have a serious project team that functions well, then it's nearly impossible."

There was also a teacher (T6) who felt that it is not his job to be police over students. He did admit that it might mean that in some cases students got a passing grade without having done anything. He also noted experiences where at the assessment students suddenly started complaining that another had not done anything after grading. He relied primarily on students informing him if someone was really not doing anything.

Several teachers (T1, T3, T6) did refer to a 'gut feeling' when trying to focus on assessing contributions.

Three teachers (T1, T3, T6) made it a topic of discussion during guidance sessions. Over and under achievement were viewed as a similar problem. One teacher noted that, if one student spent 40 hours a week on a project while only 12 is required, it would make a topic

[15] This refers to the daily stand-up that is part of the scrum methodology, teachers often only see the stand-ups during guidance sessions.

of discussion with the group.

In cases of serious under performing two teachers noted that they would first have a private chat with a student to see if there were external reasons:

"Make it a topic of discussion. Underperforming and overperforming is according to some kind of baseline. Based the number of studypoints[16] *we as a study have an opinion on how much effort there should be in the project. But I believe it's always important to discuss this with students."* (T3)

Lastly two teachers noted they did take action during the grading. With one teacher (T2), if he suspected there was a large gap, to do the following: *"I have 23 points to divide, who is getting the 5?"* he claimed that the student who did the least would often raise their hand.

USAGE

Here we look at how and when the teachers used the tool during project guidance. Invitations for the interviews were made in the last weeks while the experiment was running. It became clear from 4 teachers that they had not used the tool, following which I sent out an email requesting they open the tool with students and look at it in the last two weeks of the project.

Two teachers (T4, T5) used the tool from week 2 when students had produced and committed code, looking at it with students during guidance. One teacher (T3) did look at it over the course, but did not look at it with student until the last week. Two teachers (T2, T6) only really used it in the last week of the course after en email was sent requesting the use it. One teacher (T1) only really used it once to gain information on a group he did not formerly have.

Only one teacher (T4) looked at the tool five minutes before looking at it with the group. All the other teachers opened it with the students present without looking at it beforehand.

All teachers stated that when looking at the metrics with students they tended to look at it and ask questions. Asking students to explain the graphs they were seeing:

"I open it during the session and then share my screen. And then yes, I ask the question 'what do you see? do you notice anything? Oh apparently you have modified a lot of code, is that accurate?' "(T2)

One teacher (T5) did say he used the tool to specifically point out to a student that they were not contributing enough to the project. He also discovered a student who had not committed any code at all about halfway into the project, this was explained by the student using Dropbox to send code to fellow students. T5 attributed early discovery to the tool.

In Figure 6 we can see what students reported in terms of teacher usage. Given that 4 of the 6 teachers did not use the tool frequently by their own admission, it makes sense for a large number to state the teacher only used it sometimes.

EVALUATION AND FUTURE USE

All teachers voiced positive experience to the tool, noting it gave something to discuss with students. Though not all metrics were clear to him, one teacher (T2) noted it as being a great addition to using Gitlab. Being able to get a quick overview of the project was considered the biggest positive. It was noted that at the very least this also created discussion

[16] Referencing European Credits

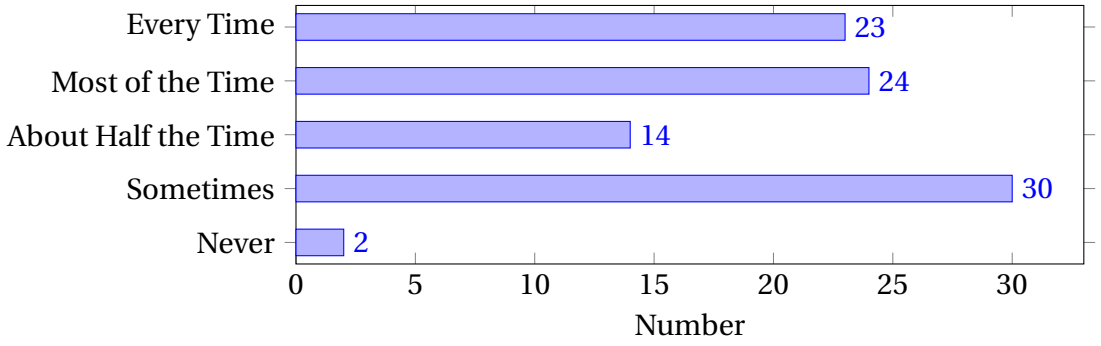

Figure 6: Student survey answers to the question: The metrics generated by the tool were discussed by the teacher during guidance sessions

among teachers:.

"Yes, I see value in it. It makes me happy. I do believe we should do something with this in our courses. Though I find it difficult that we have a dashboard of metrics while students don't know why those metrics are important." (T3)

Only one teacher (T5) considered this an assistance in grading, while the other teachers were against using this in any form of grading. A teacher (T3) did point out that it can help focus on a student to ask specific questions:

"I noticed it was easier. It was easier to identify those problem areas and it was easier to start a conversation about something like cyclomatic complexity."

The only change in their guiding approach teachers that noted was being able to ask more directed questions during sessions. Also having discussions with students about certain concepts. However, teachers in general did not feel they changed their approach to guidance significantly.

All teachers noted wanting to use this tool in future projects. All agree that it should fill a supporting role, to allow discussion with students. One teacher expressed preference for examining it with students per sprint, instead of per week. This same teacher would also ideally have the tool visible on a large screen with each group in a classroom.

There was some critique regarding the tool. Two teachers (T1, T6) stated that, as a teacher, you need to have a good story about why teachers care about these metrics. Two other teachers (T3, T4) did not fully trust the metrics, as they did not fully grok[17] how the metrics were calculated.

"Currently they are pure statistics, pure numbers. And I find that difficult because if a student then says because a b c x y z, I still have to go through the repository to verify if what they said makes sense." (T3)

"An explanation does not completely help. I want to understand it completely before I enter into a discussion with students" (T4)

5.2. RQ2: Student Response

How do students perceive the usage of tools monitoring contributions?

While the main question focuses on how this tool can help teachers in guiding students, it is interesting to observe how students respond to being monitored in this manner. First

[17] Teachers stated that they grasped and understood where they metrics came from, but this was not enough for them. https://en.wikipedia.org/wiki/Grok

teachers give their observations from guidance sessions, then students give their responses and also consider future use of this method.

According to the Teachers

In this section we look at how teachers observed student behavior, reactions, and whether they should have access or not.

Overall teachers stated that they believed students reacted positively. Reactions according to teachers ranged from it being interesting, to curious how these metrics could be generated. Two (T1, T3) stated that they believed that most students are quite aware of how much they contribute:

"Their reaction was very clear, namely that that which the tool displayed was the reality. They completely agreed with certain aspects of one student, who confirmed the information."

One teacher (T4) did note that some students were a bit shocked at the fact they were being tracked:

"I do have the feeling they were surprised. "Hey what's this? We're being monitored in some kind of graphical way." They know, but I don't think they were aware of how much. They did try to understand what was displayed and which point I was trying to make."

Interest seemed to primarily focus on the quality metrics like Complexity over the volume metrics.

None of the teachers could confidently say that they saw changed behavior with students due to the group being monitored.

Four teachers talked about giving students access to the tool, though one (T3) argued that perhaps students should only see quality metrics, leaving the quantity metrics to teachers. Another (T4) worried that giving access to the tool might cause students to artificially boost numbers:

"Look, eventually we look at the distribution. And it doesn't all have to be the same but someone has to do a contribution and it it would be nice if it was reasonably the same. But if there are some differences we can still live with that, because we do an interpretation. They can't do that themselves that well. Because they don't see the goal we are trying achieve with this."

According to the Students

On their own students who hardly saw the tool also didn't really discuss it in the group (see Figure 7, though two students claim they did. Of the students that saw the tool frequently 18 claimed to have discussed the tool most of the time or more.

Figure 8 shows how often students used the metrics to inform division of their work. Only two students who sometimes saw the tool claimed they used it most of the time to decide group task division. While 14 claimed to have used it most of the time or more to divide the tasks.

In the open question: "How did the metrics generated by the tool affect the way you worked on this project?", multiple students (36, 40%) stated that there was no change in their behavior during this project. It should however be noted that 23 of those were from students where the teacher discussed the tool half the time or less. For example, one student stated:

"In our case not much. Everyone did about the same amount of work. But otherwise you could naturally approach someone on the fact they did nothing to little work for a certain

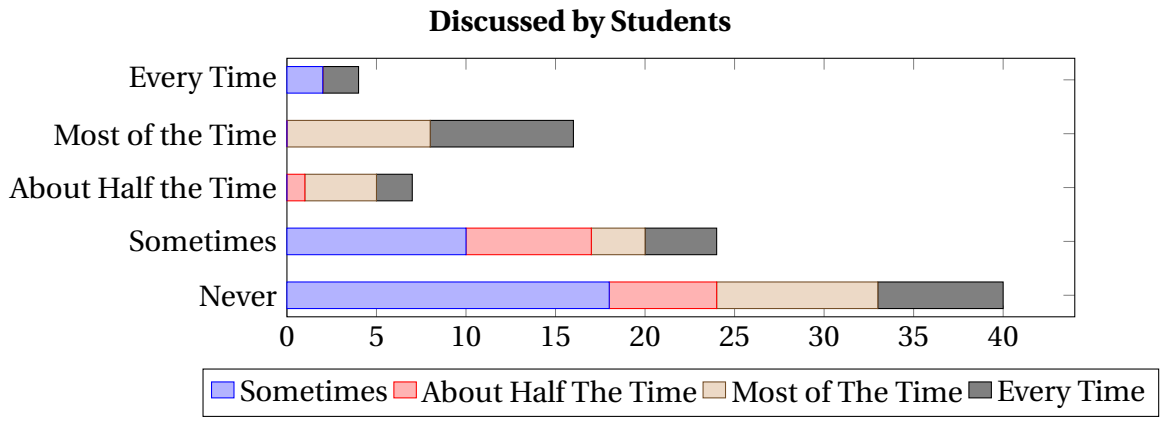

Figure 7: Student survey answers to the question: The group discussed the metrics generated by the tool on their own, split according to how often the teacher showed it to them (Figure 6).

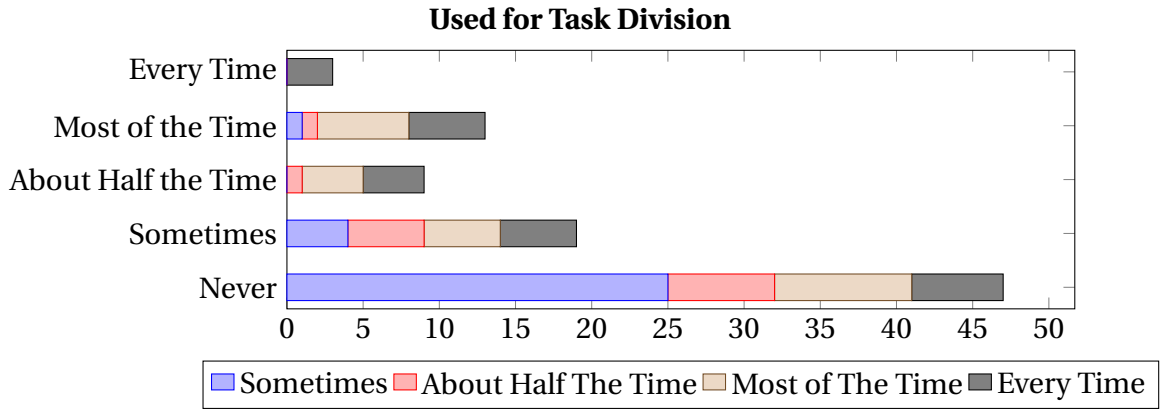

Figure 8: Student survey answers to the question: The metrics generated by the tool were used to decide on task division within the group, split according to how often the teacher showed it to them (Figure 6).

project." stated one student.

Another student responded: *"Honestly not much. I did find it useful to know. My contribution to the project matched the data that was being presented."*

There were a few students (9) who stated they started to work harder during this project, replying for example:

"It was stimulating to see how much code you had written compared to others. This caused one to be inclined to do more if another has a bit more."

Some students (9) claimed it impacted how division of work was changed due to this project.

"We got a really good idea of the division of roles within our group. After we saw this [the tool], we dropped this division of roles and started do a bit of everything within the group."

There was also a group of students (10) that merely reflected on the progress of the team being visible. In general this was received as a positive thing.

"Before looking at the tool we already had an idea of division of tasks in our group. Our expectations generally matched the tool so we were not surprised. It was useful to still see an overview of the division of tasks."

There were a few negative responses the question in regards to the changes to their behavior.

"Personally not so well, because the guiding teacher assigned a lot of value to the statistics."

"That I became demotivated because I thought I did a lot but eventually compared to the rest I did not do a lot."

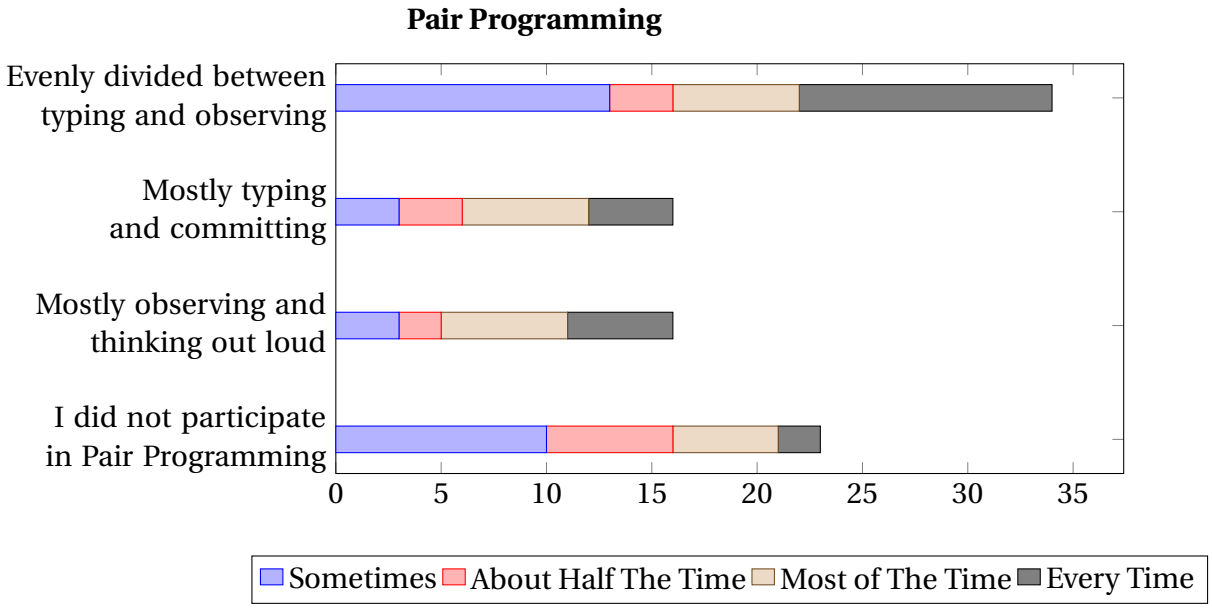

Figure 9: Student survey answers to the question: Pair Programming Behavior, split according to how often the teacher showed it to them (Figure 6).

Pair programming is something several students did during the project. Though not required or formally introduced to students, it is common for them to do this, especially since a lot of the work was done online with students sharing screens. In Figure 9 we can see that the majority claimed they divided the work evenly.

In the open questions, students pointed out that pair programming could cause skewed results:

"It is a shame that when you do pair programming you don't see it, but I also think it would be difficult to make that visible."

Some students stated they would let others simply type more to boost their numbers, for example:

"At some point we let another teammate type more during our pair sessions, so that the division would be better at the end."

Evaluation and Future Use

In the open question: "How should we use this tool in future projects? Should we add or change something in the tool or the way we use it?", students that saw the tool frequently expressed that no changes were needed to it and should be used in future projects.

"In a group, there will always be someone who works more than the other, therefore I think this tool could help detect the people who are less interested in the project and maybe give them an impulse into working more"

"Discuss with the students about how they feel when they see the graphs, and ask about low values, but don't use this in grading, as there are many ways to help the group during a project without using Git."

Students that only saw the tool in the last two weeks commonly expressed the tool should have been shown directly at the start. They also wanted to have access to the tool.

"It is probably better to use the tool from the first sprint, so potential problems in groups will come to light sooner. That way the tool would have more effect on the way groups work."

There was only one student who compared the use of the tool to a country that monitors its inhabitants, but in the same sentence also considered it fine. It is unclear if the student was aware at the time of the project, that Git stores all meta data regarding commits. The tool only changes the way this data is displayed.

5.3. RQ3: METRICS

How are utilized metrics perceived by teachers and students?

	Not		**Kind of**		**Fairly**		**Very**	
	Clear	*Accurate*	*Clear*	*Accurate*	*Clear*	*Accurate*	*Clear*	*Accurate*
Lines of Code	1	3	6	15	41	48	43	25
Comments	3	5	18	17	36	43	34	26
Documentation	1	3	16	19	44	41	30	28
GitBlame	3	6	26	27	37	42	25	16
Complexity	13	2	29	27	32	45	17	17
Method Size	3	1	26	14	37	52	25	21
Time of Commit	4	1	16	12	32	51	37	27

Table 4: Clarity and Accuracy of each metrics as observed by students.

Focusing on the second version each metric that was presented there, except the commit overview (Figure 5). For each metric answers from the interviews are used combined with answers on the survey by students. Students were asked to rank clarity and accuracy of each metric.

Table 4 displays the total numbers of responses by students divided over clarity and accuracy. In the following sections we look at each metric in detail supported by comments made by teachers about these metrics.

LINES OF CODE

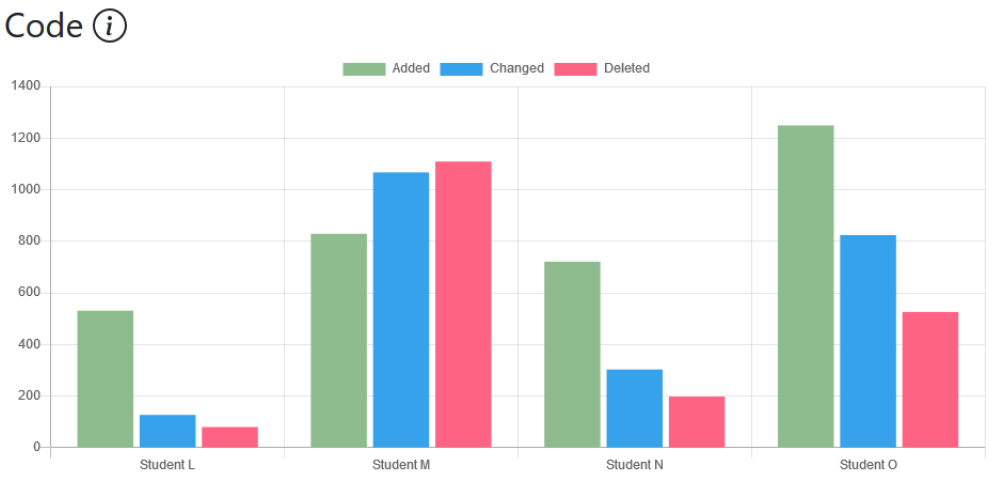

Figure 10: Example of Lines of Code metric display

All teachers mentioned looking at this metric with students. An example of how it appeared to them can be seen in Figure 10. They also all expressed their worries that students might try and inflate these numbers. Only one teacher (T4) had this actually happen with a group, where they added hard coded matrices for level data to their project. Another teacher (T5) had an instance where a student copied the supplied library resulting in not really paying attention to that student during the project, assuming erroneously they had produced enough.

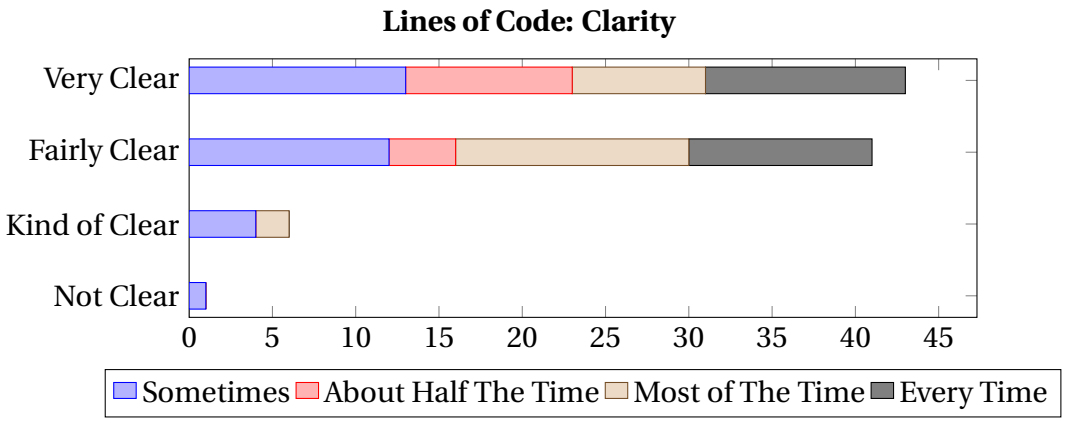

Figure 11: Student survey answers to the question: How clear were Lines of Code, split according to how often the teacher showed it to them (Figure 6).

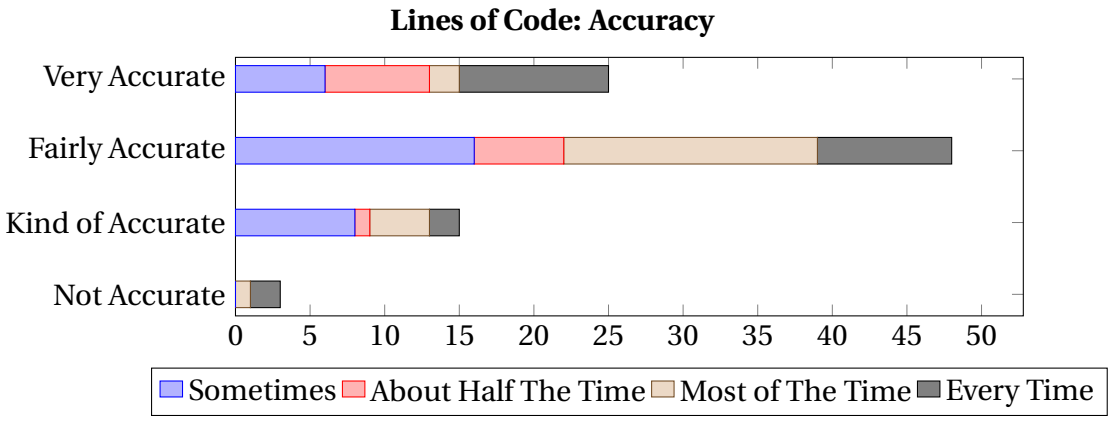

Figure 12: Student survey answers to the question: How accurate were Lines of Code according to students, split according to how often the teacher showed it to them (Figure 6).

In terms of critique, one teacher (T1) did point out that students could have written a lot of code and made all the wrong decisions, resulting in a lot of rework. In similar style another teacher (T3) argued that a student might work on one small complex thing, while another is working on a lot of simple things. He also suggested that perhaps for this metric to work, an expected amount of code should be given for students to benchmark against. Lastly, one teacher (T4) asserted that the less lines of code written, the better. But he also admitted that perhaps students at this level are not ready for that yet:

"I find lines of code a supporting factor. In my head there is a sort of balance, if you are busy with complex challenges that are in your project. So you pulled the most difficult pieces towards you and then it's logical you're not working a wider scope. Then you work on a wider scope, so you do superficial things and never make it hard on yourself. Then I expect that you do more. And students should find a balance in that."

Students listed lines of code to be the very clear to understand (Figure 11). Similarly, they noted them to be fairly accurate (Figure 12). In Figure 13 we can see that only students who observed during pair programming settings stated the metrics were not accurate.

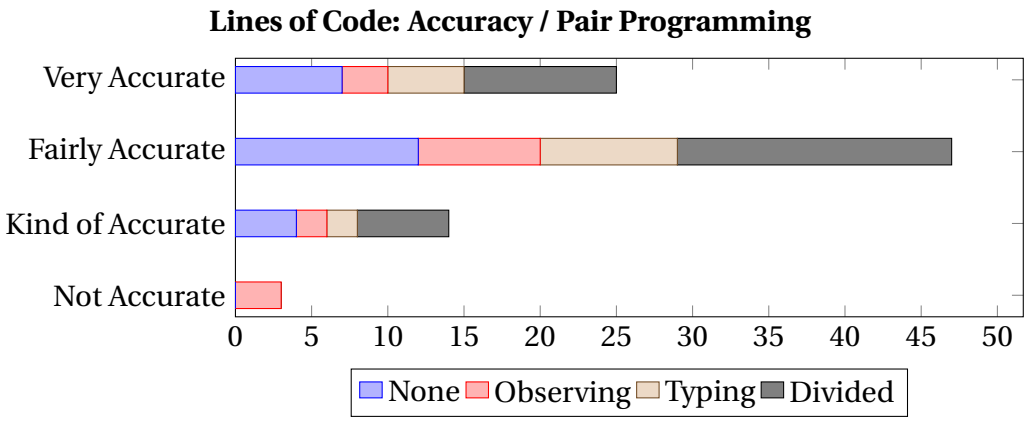

Figure 13: Student survey answers to the question: How accurate were Lines of Code according to students, split according to Pair Programming activity (Figure 9).

COMMENTS/JAVADOC

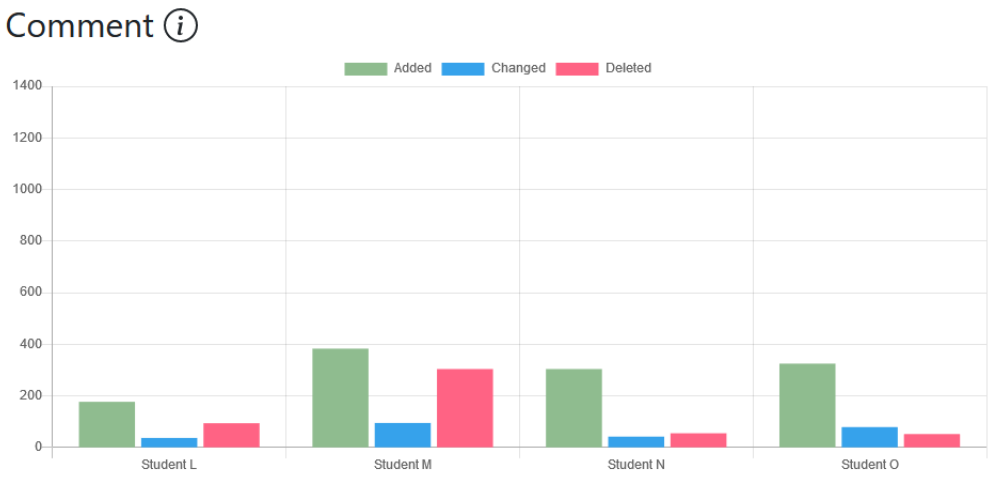

Figure 14: Example of Comment metric display

During the experiment this was commonly referred to as 'comments' though it included JavaDoc as well in this context.

Only three teachers (T2, T3, T6) mentioned looking at comments (Figure 14) but nothing in detail. With one teacher (T3) primarily wanting to know if students were doing proper JavaDoc.

Clarity (Figure 15) and accuracy (16) had a few more responses claiming both not clear and not accurate, though overall the results are similar to that of lines of code.

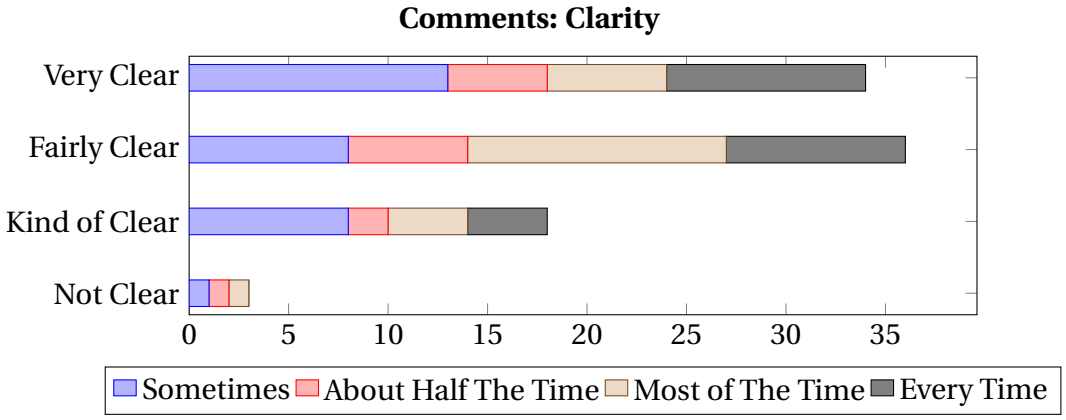

Figure 15: Student survey answers to the question: How clear were Comments, split according to how often the teacher showed it to them (Figure 6).

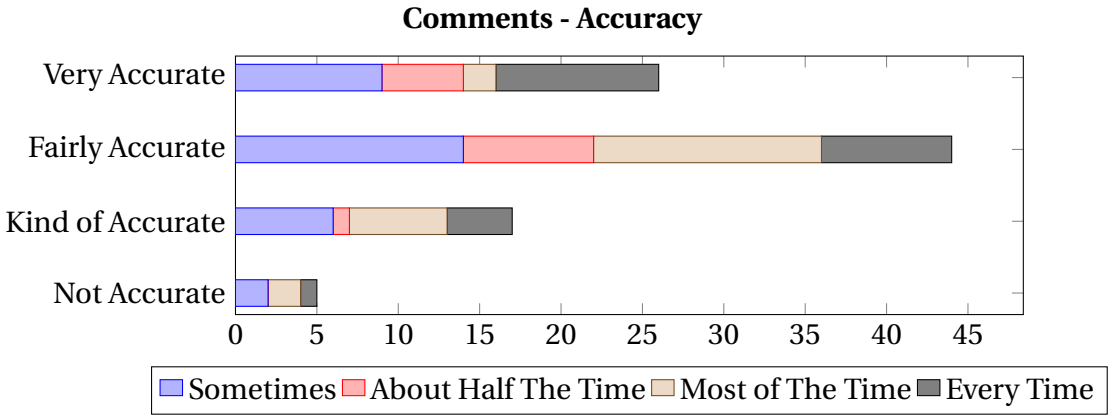

Figure 16: Student survey answers to the question: How accurate were Comments according to students, split according to how often the teacher showed it to them (Figure 6).

DOCUMENTATION

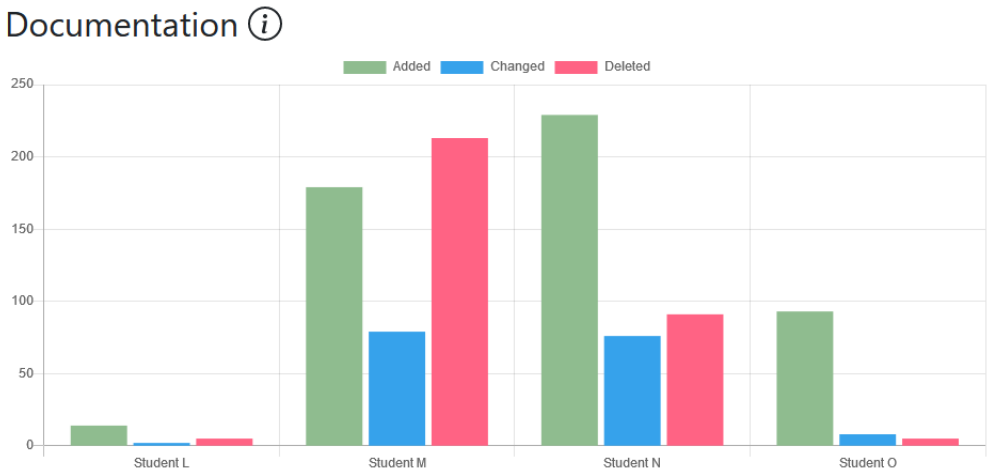

Figure 17: Example of Documentation metric display

Only two teachers (T2, T6) mentioned documentation (Figure 17) and in both cases stated that students wrote it somewhere else and then one added it, or only two really worked on the documentation.

"You'll see that in a team there are only one or two that worry about that, and maybe the others worked on it or make a few sketches, but you don't see that. So maybe the least interesting in a class, I'm not sure."

Clarity was rated fairly clear by students (Figure 18). Despite observations by teachers regarding how it often happens, student still rated the accuracy fairly high (Figure 19).

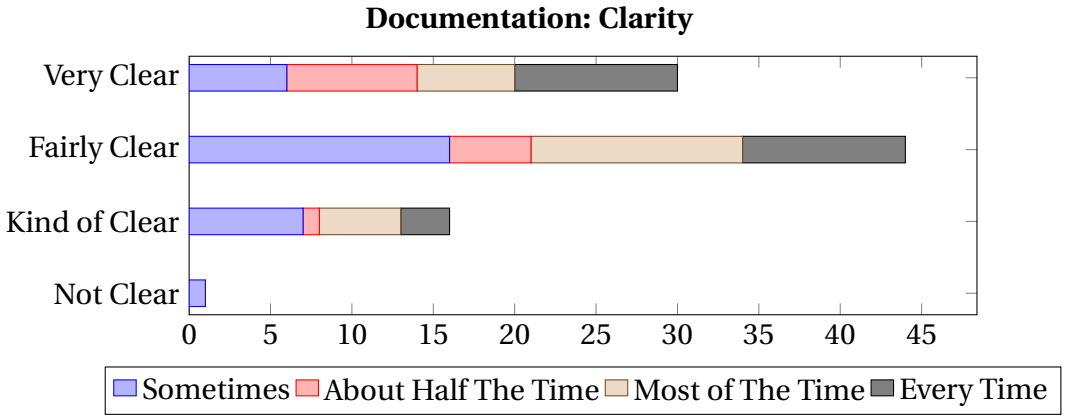

Figure 18: Student survey answers to the question: How clear were Documentation, split according to how often the teacher showed it to them (Figure 6).

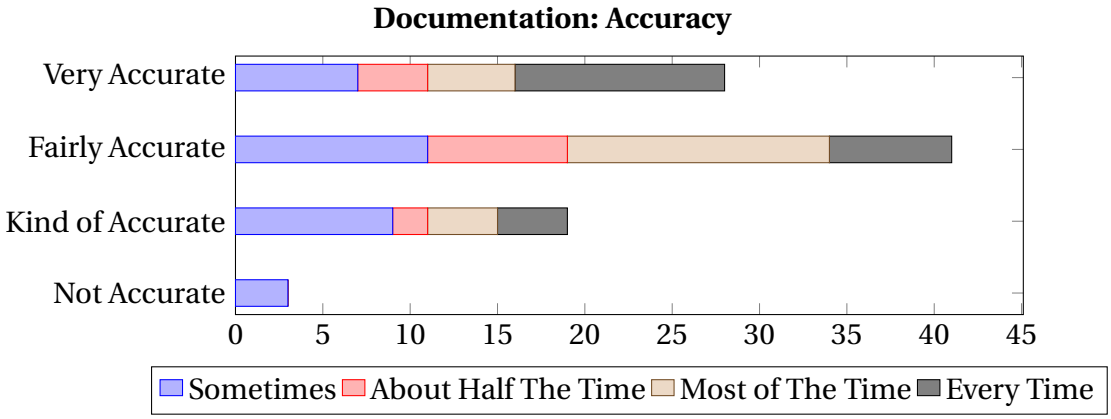

Figure 19: Student survey answers to the question: How accurate were Documentation according to students, split according to how often the teacher showed it to them (Figure 6).

GIT BLAME

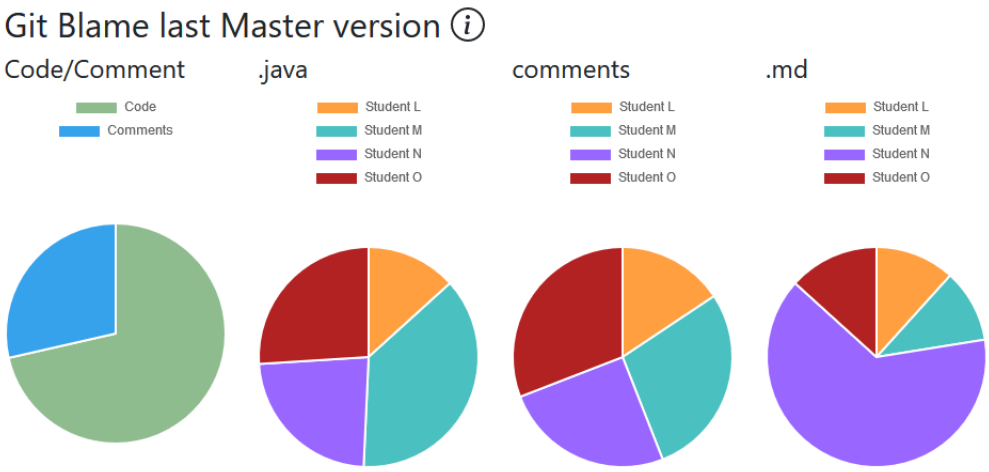

Figure 20: Example of Git Blame metric display

Two teachers (T2, T6) stated not being familiar with Git Blame at all, an example of how it was presented can be seen in Figure 20. The other teachers were in general no entirely clear on what the chart meant. This chart did not change during date range changes which sometimes led to confusion. One teacher (T3) would have liked to see it mapped out over time as well.

"I believe I looked at it once. But it did not really come up." (T5)

Clarity (Figure 21) is similar to Lines of Code. Most students only found it fairly accurate (Figure 22).

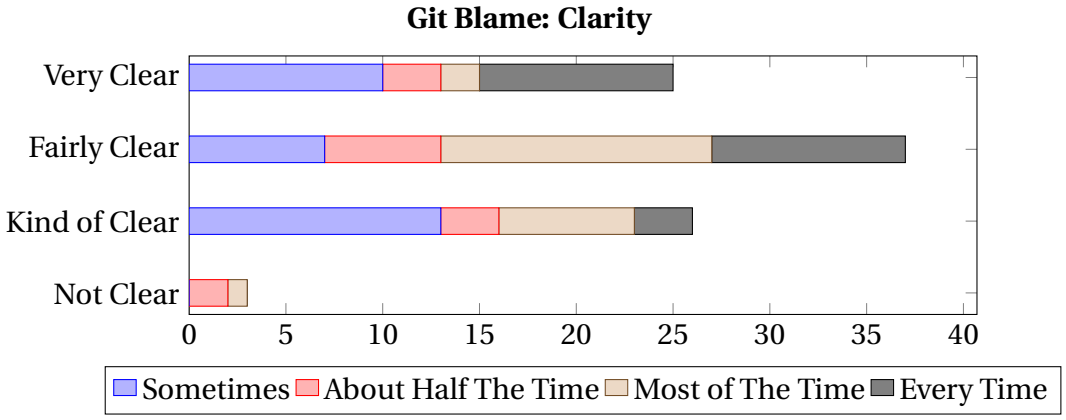

Figure 21: Student survey answers to the question: How clear were Git Blame, split according to how often the teacher showed it to them (Figure 6).

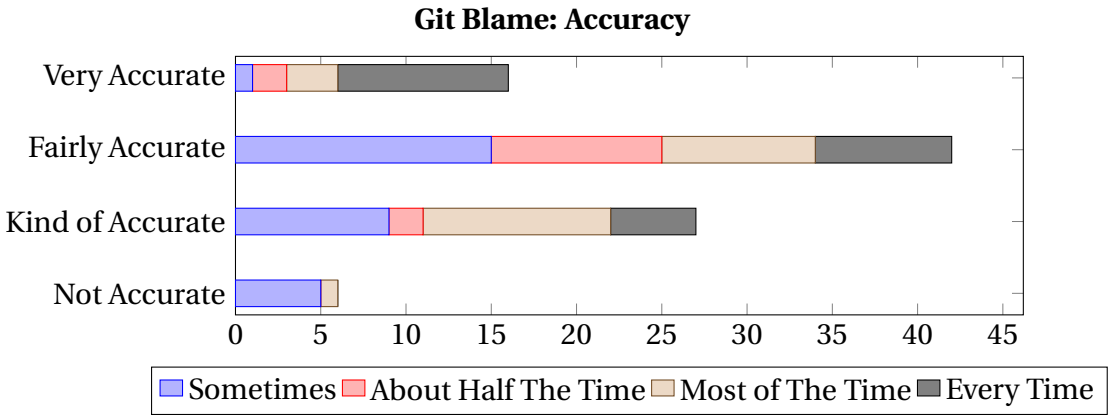

Figure 22: Student survey answers to the question: How accurate were Git Blame according to students, split according to how often the teacher showed it to them (Figure 6).

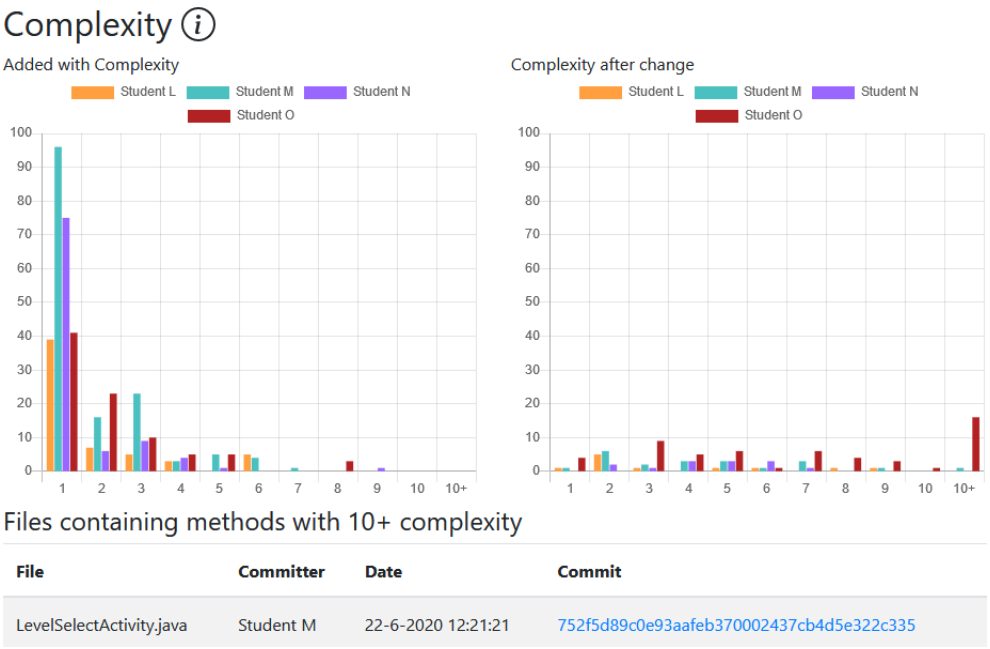

Figure 23: Example of Complexity metric display

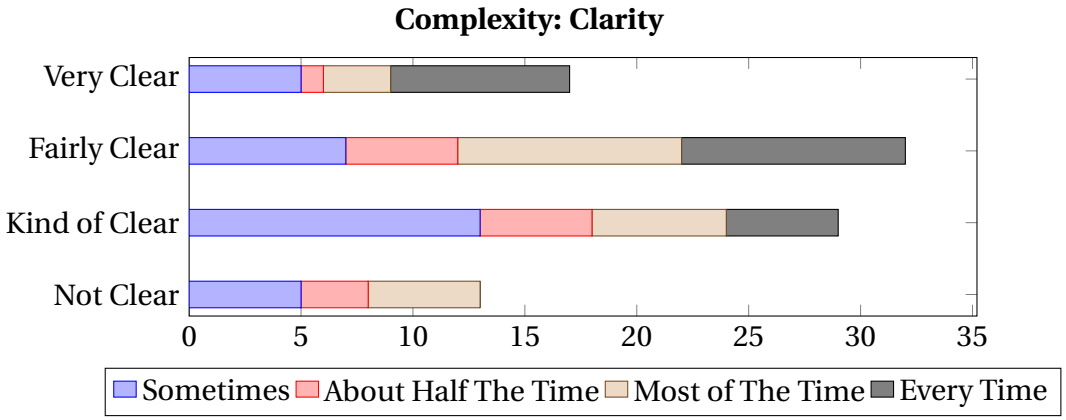

Figure 24: Student survey answers to the question: How clear were Complexity, split according to how often the teacher showed it to them (Figure 6).

Most teachers stated confusion or lack of clarity on what this graph represented, an example of which can be seen in Figure 23. Two teachers (T2, T6) stated not being familiar with the concepts of complexity. One teacher (T4) had previous experience with SonarQube, where he was familiar with how the metrics were generated, but with this tool he did not fully understand how the metrics were generated and thus glossed over it. Another teacher (T3) argued that it was too early for students to understand why we consider this relevant, though he did state it could be used to introduce the concept of complexity. Another teacher (T5) used it as explanation as he stated students were learned about it in another class:

"If I say 'hey your code here is too complex, you should break this up and write it more efficient' is something else. In year 1 that does not land. Year 1 has no idea what I'm talking

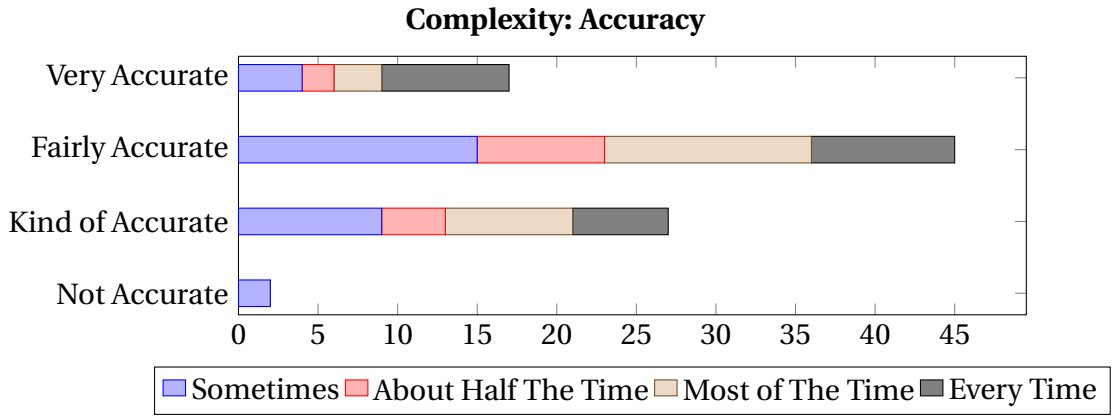

Figure 25: Student survey answers to the question: How accurate were Complexity according to students, split according to how often the teacher showed it to them (Figure 6).

about. And in that moment you actually want to use that tool as an introduction method to introduce the subject. So I tried it with my group, explain complexity based on the data in the tool."

One teacher (T6) that was initially unfamiliar with the metric, once explained did point out that five sequential If statements would result in a similar complexity as five nested If statements. According to him five nested would be less desirable.

Teachers stated that they used the link feature to locate methods with a high complexity.

A suggestion that was made by a teacher was for complexity to also be displayed for the project rather than the current format[18]

Students rated clarity quite a bit lower than other metrics (Figure 24), though seem to find it reasonably accurate (Figure 25).

[18]Which is based on per person collected over commits in the range.

METHOD SIZE

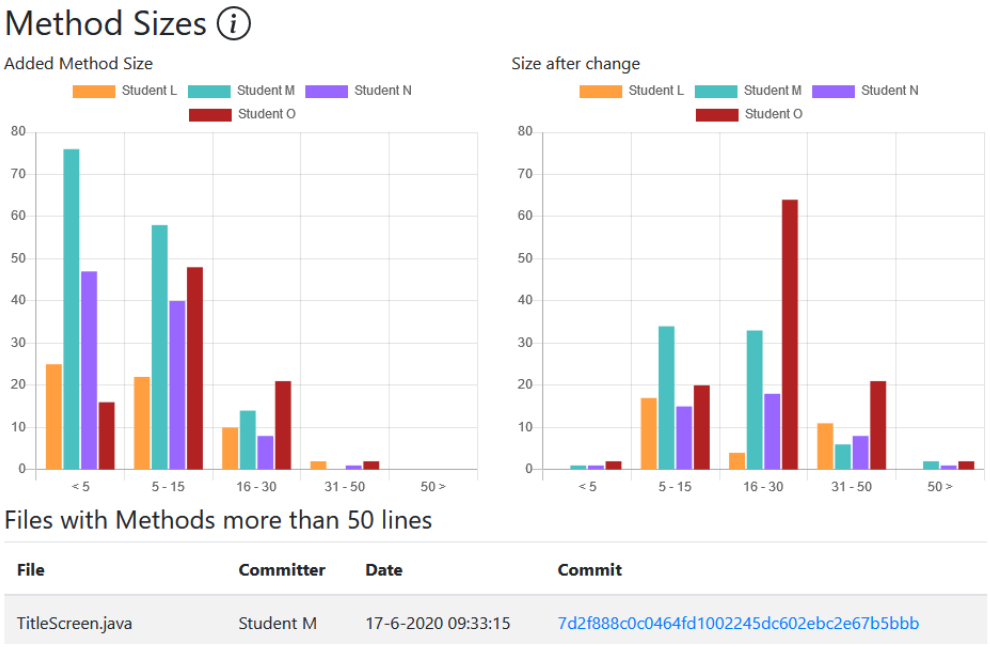

Figure 26: Example of Method Size metric display

Four teachers (T3, T4, T5, T6) mentioned using and discussing the method size metric with students, an example of which can be seen in Figure 26. They all stated having discussions with students regarding method size and why large methods might not be a good idea. There was wonder about the chosen intervals in which method size was divided to (T3), but being able to quickly locate large methods was considered convenient:

"So method length, you display it now as less than 5, 5-10, etc and 50+. But we most certainly have an opinion about what is good and what not. 50+ is a reason why you categorize it as 50+. Because your thoughts, and mine too, might be that that is a bit much. 50 lines in a method."

One teacher (T6) pointed out that it does say something, but he was not sure it was always the right thing. If someone has a large number of small methods, they might have just generated a bunch of getters and setters. Similarly, someone who wrote a few large methods could be doing interesting work, or is not a very good programmer.

Students rated method size reasonably clear (Figure 27) and fairly accurate (Figure 28).

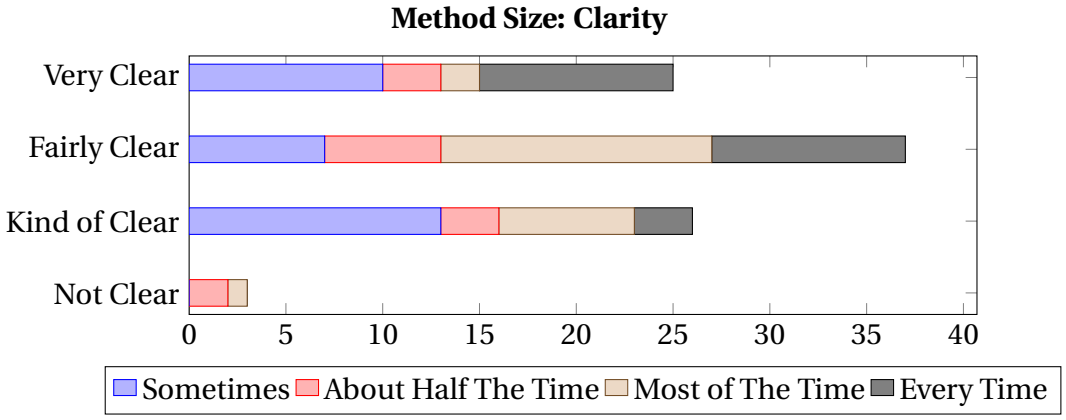

Figure 27: Student survey answers to the question: How clear were Method Size, split according to how often the teacher showed it to them (Figure 6).

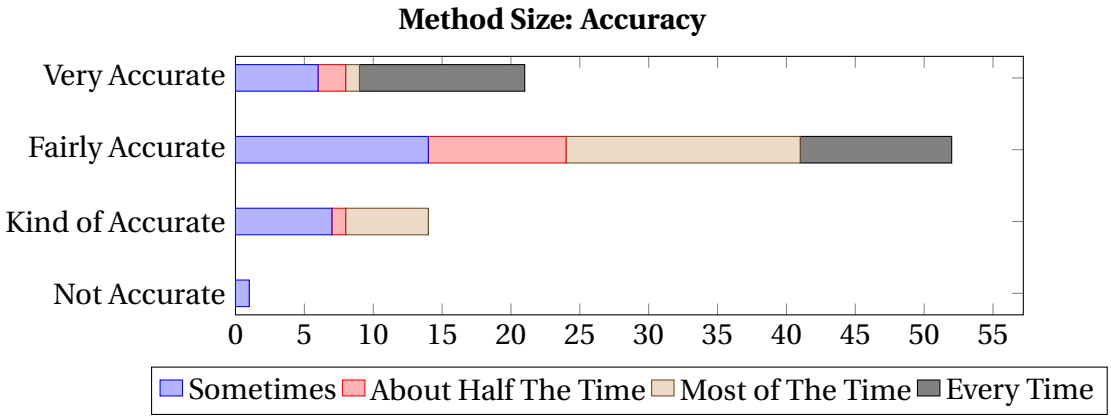

Figure 28: Student survey answers to the question: How accurate were Method Size according to students, split according to how often the teacher showed it to them (Figure 6).

COMMIT TIME OF DAY

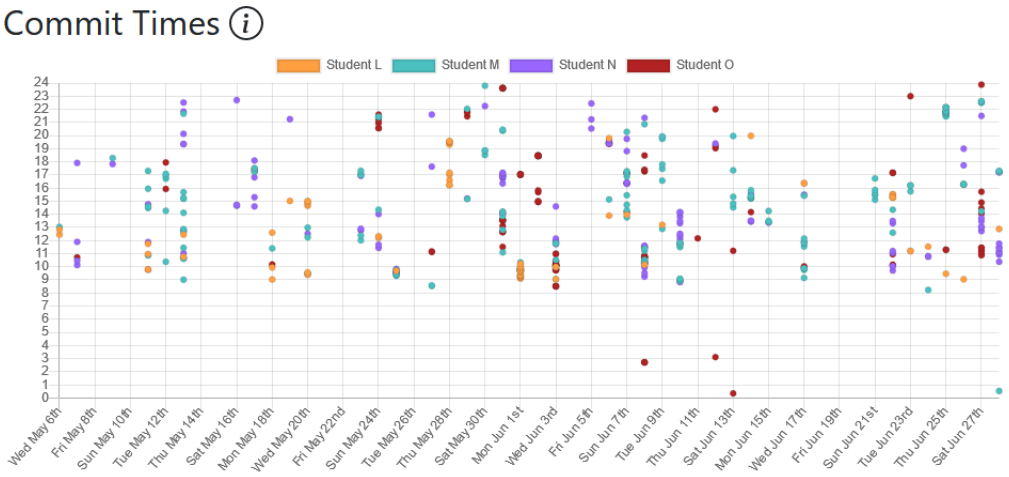

Figure 29: Example of Commit Time of Day display

All teachers but one stated having looked at this metric, as seen in Figure 29. Being able to see if people were committing together as groups being stated as useful. Though only three (T2, T3, T6) discussed it further during the interviews, two of which noted amusement with students committing during the night.

"and that was a nice overview, the commit times. That you can clearly see if people are committing as a group. I mentioned that to a bunch of teams that we liked to see that. But with some it wasn't a lot outside of class." (T2)

"Yes that is a fun one. And student found that too. If it's that useful... but we did have fun about who was committing in the middle of the night. That sort of stuff. But it is very readable in this form." (T6)

Students found this the most clear metric (Figure 30), though most students found it only fairly accurate (Figure 31).

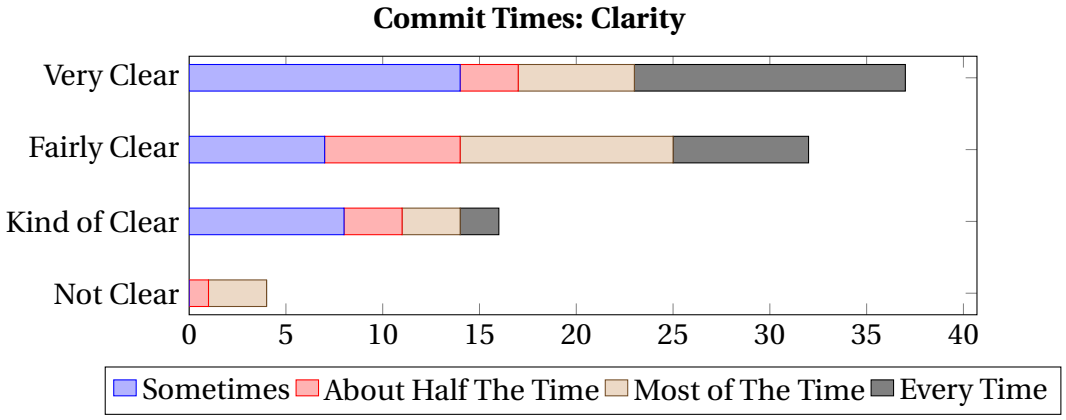

Figure 30: Student survey answers to the question: How clear were Time of Commit, split according to how often the teacher showed it to them (Figure 6).

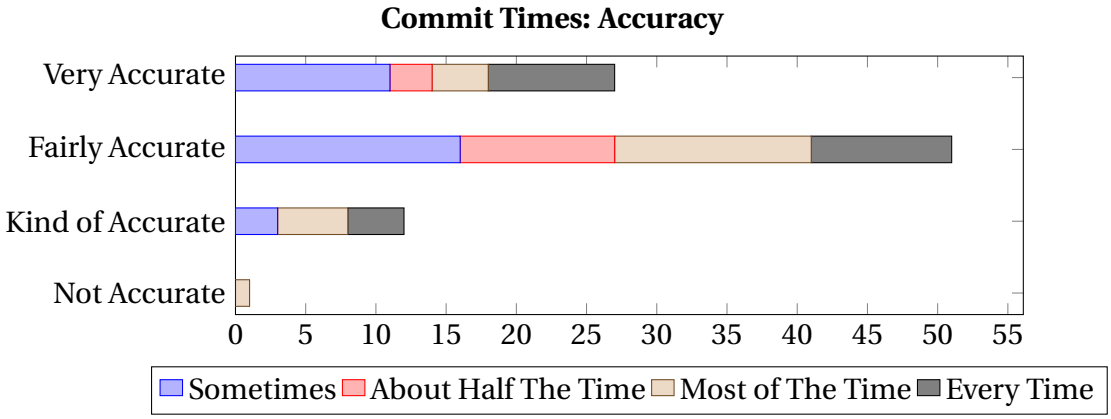

Figure 31: Student survey answers to the question: How accurate were Time of Commit according to students, split according to how often the teacher showed it to them (Figure 6).

Missing Metrics

The following metrics were mentioned during interviews that teachers would have been interested in:

1. Code Quality Metrics based on ISO norm (T1)
2. Test Coverage (T1, T3, T5)
3. Path coverage (T1)
4. Code Reviews (T4)
5. Code Duplication (T5)

The interview did not go into detail on their representation or on how to calculate these.

A few students did argue that editing media files is also part of the project and should also be represented:

"I worked on pixel art images for our game, but that does not count as code or anything. So maybe a graph how many media files people have added?"

Representation

Several teachers stated wanting a less cluttered view. Also a separation of quality and quantity metrics was mentioned. Clearer explanations of the metrics was also requested as the current information panels were found to be very technical (T6).

Two teachers (T2, T3) mentioned wanting to be able to quickly go to the code that the metrics represented. One example given was that you can see they modified a lot of code, but what code is an unknown. Another teacher stated that he wanted to click on each graph to be taken to actual code to examine it.

Some students similarly stated, in the question regarding future changes, they would like more clarity in the design.

"Maybe the interface/design of the page to look more clearly and easier to see the data. "

6

Discussion

Even though this experiment took place under less than ideal circumstances with everyone working at home due to Covid-19, there are interesting results from the interviews and survey.

Given that an experiment like this has not yet been done (as far as the researcher could find) the results are very promising in terms of this approach having a positive effect. In general both teachers and students responded positively to the concept of these metrics and to being able to see contributions.

It is interesting to observe that given the strange and hectic circumstances the teachers that have more experience tended to not use the tool in classes. It should also be noted that if everyone had been working from the physical locations, the researcher would have encountered them and asked about the tool. This did not happen due to work at home circumstances.

In hindsight it would have been a better idea to devise clear usage instructions for this tool for teachers. However, how to use a tool like this was somewhat of an unknown going into this experiment. Based on the results it is now obvious that clear instructions should be imparted on teachers when using a tool like this. Willingness to use this tool by the teachers and awareness of students (given that their course page contained detailed information) were underestimated during this experiment.

Students in general seemed to respond to this method positively, possibly because they did not have to rat out others to their teacher. This is in line with prior work of Colbeck et al. stating that students do not enjoy ratting out fellow students [6]. This tool takes away this barrier.

There were also students who clearly enjoyed seeing the numbers go up and even getting a bit competitive. This is in line with concepts of gamification similar to what Dubois and Tamburrelli found [9].

A reasonably large amount of metrics were used during this experiment. This was caused in part due to readily available off the shelf metrics that could be generated. The other part was teachers having different requests. In hindsight a more critical selection of metrics could have helped focus the responses.

Conversely, some metrics such as the estimated time spent suggested by Parizi et al.[22], turned out to be not as useful during a project such as this. Primarily because teachers were not interested, but also because students had not yet developed a consistent habit of

committing code. The researcher encountered various students that would commit when something was done rather than at the end of the day.

Despite some critique from teachers, they clearly indicated wanting to use this tool further in the future to support asking directed questions of students and to help explain programming concepts. The overall reaction from teachers was a positive one.

There is a question if this specific tool was needed for this research. Other tools such as SonarQube[19] exist, and websites like Github and Gitlab offer some degree of insights. However, SonarQube focuses primarily on software quality rather than on personal contributions. Similarly a tool called GitInspector[20] which seems only only referenced in a bachelor thesis[21], has some of the same metrics but not all. Most notably both require some work in use and setup which could introduce another barrier of entry to teachers and students to use it. The tool as used here was fairly low barrier to use and deploy and allowed for fairly custom tailoring of displaying the metrics. Though it could be argued that any tool displaying metrics could be used in a method such as this, careful consideration of how many metrics is however prudent.

In his own class the researcher noticed a more balanced division of work with students being honest about how much they did. Students were also actively planning based on sprints. For instance two students doing more in the early sprint, but planning for complex task for the others in the later sprints. In another group there were two who had far more programming experience due to previous schooling, and consequently they divided their work so the students newer to programming did more.

This can be potentially be attributed to the following factors:

- Students are made aware they are required to code.
- Students are aware that teachers can observe their contributions in detail.

These factors create a situation of external pressure that potentially prevents them from trying to coast by unnoticed. This can also be attributed to the Hawthorne effect, as people being monitored will change their behavior.

An observation that was common during the interviews was that it is important to constantly explain to students what the purpose is, why and how metrics are gained, and that it will not directly affect grade. This is consistent with the findings by Sudhakar et al. in that when metrics become a goal it impacts developer behavior negatively [28]. Obviously if a student does absolutely nothing the tool could be used to point this out to them to adjust their behavior given that students are graded on their capacity to code, so there has to be something in the project teacher can examine to judge their ability to code.

LIMITATIONS

First and foremost, this research took place during the height of the lockdown period of 2020 caused by Covid-19. As such all involved teachers and students were working from home per protocols put in place by government and the institution. This on its own gives a large impact on how teachers and students worked during this experiment.

[19] https://www.sonarqube.org/
[20] https://github.com/ejwa/gitinspector
[21] https://digitalcommons.wpi.edu/mqp-all/3210/

Other limitations that can be identified are, firstly, that this study was performed at one school for one project. Results might vary in different schools, countries, cultures, assignment setups, and years of experience of students. The study also did not differentiate between students following the international track and students following the regular Dutch track.

Second, the internal validity can be threatened in the interviews due to the teachers also being co-workers of the researcher. While they seemed honest about the tool and the research, they might still have preferred being friendlier in their responses. In processing the data there is a risk of how the interviews were labeled. We followed a data driven approach, where opinions and exceptions were also explained properly in the research. The focus was maintained on the thematic approach of each subject.

Third, the survey was suggested to be filled out by the teachers during class hours. This means that there was a measure of compulsion in filling it out, which can affect how honest students were in filling it out. Moreover, the class guided by the researcher also filled it out, which might have colored their opinion positively or negatively based on how they viewed the researcher.

7

Conclusions

Here we will summarize the answers to the research questions based on the results and discussion.

RQ1: How do teachers utilize these metrics within their guidance?

The primary approach by teachers was discussing the metrics with students and asking them to explain the metrics that are being presented. They also used it as way of explaining concepts of software quality using examples from the report. There was a strong desire to use a system such as this in future projects as it saved time.

RQ2: How do students perceive the usage of tools monitoring contributions?

The overall response of students was positive. Students who saw the tool very late in the project stated they would have liked to have seen it sooner. General response was being able to see who did how much was seen as positive.

RQ3: How are utilized metrics perceived by teachers and students?

Lines of code, complexity, and Method Size yielded the most response among both teachers and students. Lines of Code gave a very clear indication of productivity, while complexity and method size introduced quality concepts to students. These metrics also allowed teachers to ask more directed questions during guidance sessions.

In the context of group programming assignments in higher education, which metrics and visualizations are suitable and useful in assisting monitoring and guiding individuals in a group by teachers?

While a reasonable number of metrics were examined and tested, it was clear that lines of code due to its simplicity is easy to understand and process. Lines of code with complexity and method size allowed teachers to compare the 'volume' with the 'quality' of the code being delivered.

The Hawthorn effect creates a potential positive effect of students being more honest in their productivity, due to student knowledge of being monitored.

However, it was very clear from responses that methods such as this should never be used directly in grading. It should always require a teacher interpreting and discussing the metrics with the students.

Future Work

This experiment yields several opportunities for future work. While various metrics were examined it might be interesting to look more in detail in which metrics have a better effect.

Especially for Lines of Code, Complexity, and Method Size.

As noted in the discussion, it is possible that similar tooling might yield similar results. This is something that could also be explored, though care should be taken is barriers of usage in such a research.

Student behavior could perhaps be examined in more detail based on the presence of a tool such as this. Similarly whether or not students should have access to the quantity metrics is also something worth exploring.

The effect of the use of a monitoring tool in groups working on software projects and the optional ways to us it in classes can also be researched in more depth.

REFERENCES

[1] Tim Blok and Ansgar Fehnker. “Automated program analysis for novice programmers”. In: *Third International Conference on Higher Education Advances* (2017) (cit. on p. 5).

[2] Rachel Cardell-Oliver. “How can software metrics help novice programmers?” In: *Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114*. Australian Computer Society, Inc. 2011, pp. 55–62 (cit. on pp. 5, 7).

[3] Kenneth J Chapman and Stuart Van Auken. “Creating positive group project experiences: An examination of the role of the instructor on students’ perceptions of group projects”. In: *Journal of Marketing Education* 23.2 (2001), pp. 117–127 (cit. on pp. 1, 3).

[4] Jian Chen, Guoyong Qiu, Liu Yuan, Li Zhang, and Gang Lu. “Assessing teamwork performance in software engineering education: A case in a software engineering undergraduate course”. In: *2011 18th Asia-Pacific Software Engineering Conference*. IEEE. 2011, pp. 17–24 (cit. on p. 4).

[5] Shyam R Chidamber and Chris F Kemerer. “A metrics suite for object oriented design”. In: *IEEE Transactions on software engineering* 20.6 (1994), pp. 476–493 (cit. on p. 5).

[6] Carol L Colbeck, Susan E Campbell, and Stefani A Bjorklund. “Grouping in the dark: What college students learn from group projects”. In: *The Journal of Higher Education* 71.1 (2000), pp. 60–83 (cit. on pp. 1, 3, 43).

[7] Juliet Corbin and Anselm Strauss. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage publications, 2014 (cit. on p. 4).

[8] John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017 (cit. on pp. 9, 11).

[9] Daniel J Dubois and Giordano Tamburrelli. “Understanding gamification mechanisms for software development”. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM. 2013, pp. 659–662 (cit. on pp. 5, 7, 43).

[10] Eduardo Fernandes, Johnatan Oliveira, Gustavo Vale, Thanis Paiva, and Eduardo Figueiredo. “A review-based comparative study of bad smell detection tools”. In: *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. 2016, pp. 1–12 (cit. on p. 6).

[11] Yossi Gil and Gal Lalouche. “On the correlation between size and metric validity”. In: *Empirical Software Engineering* 22.5 (2017), pp. 2585–2611 (cit. on p. 5).

[12] Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. “Measuring developer contribution from software repository data”. In: *Proceedings of the 2008 international working conference on Mining software repositories*. ACM. 2008, pp. 129–132 (cit. on pp. 4, 6, 7).

[13] Jane Huffman Hayes, Timothy C Lethbridge, and Daniel Port. “Evaluating individual contribution toward group software engineering projects”. In: *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE. 2003, pp. 622–627 (cit. on p. 4).

[14] David W Johnson et al. *Cooperative Learning: Increasing College Faculty Instructional Productivity. ASHE-ERIC Higher Education Report No. 4, 1991*. ERIC, 1991 (cit. on pp. 1, 3).

[15] Jon A. Krosnick and Stanley Presser. *Handbook of Survey Research*. 2nd ed. Emerald, 2010 (cit. on p. 11).

[16] Khoa Le, Caslon Chua, and Rosalind Wang. "Mining software engineering team project work logs to generate formative assessment". In: *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*. IEEE. 2017, pp. 78–83 (cit. on p. 4).

[17] Jalerson Lima, Christoph Treude, Fernando Figueira Filho, and Uirá Kulesza. "Assessing developer contribution with repository mining-based metrics". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2015, pp. 536–540 (cit. on pp. 4, 7).

[18] Rüdiger Lincke, Jonas Lundberg, and Welf Löwe. "Comparing software metrics tools". In: *Proceedings of the 2008 international symposium on Software testing and analysis*. 2008, pp. 131–142 (cit. on pp. 6, 7).

[19] Thomas J McCabe. "A complexity measure". In: *IEEE Transactions on software Engineering* 4 (1976), pp. 308–320 (cit. on p. 5).

[20] Tri Nguyen and Caslon Chua. "Predictive tool for software team performance". In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2016, pp. 373–376 (cit. on p. 4).

[21] Benedikt Mas y Parareda and Markus Pizka. "Measuring productivity using the infamous lines of code metric". In: *Proceedings of SPACE 2007 Workshop, Nagoya, Japan*. 2007 (cit. on p. 5).

[22] Reza M Parizi, Paola Spoletini, and Amritraj Singh. "Measuring Team Members' Contributions in Software Engineering Projects using Git-driven Technology". In: *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE. 2018, pp. 1–5 (cit. on pp. 5, 7, 43).

[23] Arnold L Patton and Monica McGill. "Student portfolios and software quality metrics in computer science education". In: *Journal of Computing Sciences in Colleges* 21.4 (2006), pp. 42–48 (cit. on p. 5).

[24] David N Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. "Conditions of learning in novice programmers". In: *Journal of Educational Computing Research* 2.1 (1986), pp. 37–55 (cit. on p. 5).

[25] Jenny Rowley. "Designing and using research questionnaires". In: *Management Research Review* (2014) (cit. on p. 11).

[26] Davide Spadini, Alberto Bacchelli, and Maurício Aniche. "Pydriller: Python framework for mining software repositories". In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM. 2018, pp. 908–911 (cit. on p. 14).

[27] Margaret-Anne Storey, Thomas Zimmermann, Christian Bird, Jacek Czerwonka, Brendan Murphy, and Eirini Kalliamvakou. "Towards a Theory of Software Developer Job Satisfaction and Perceived Productivity". In: *IEEE Transactions on Software Engineering* (2019) (cit. on p. 4).

[28] Goparaju Sudhakar, Ayesha Farooq, and Sanghamitra Patnaik. “Measuring productivity of software development teams”. In: *Serbian Journal of Management* 7.1 (2012), pp. 65–75 (cit. on pp. 4, 44).

8

APPENDICES

A. ADDITIONAL IMAGES OF TOOL

The following page display a few more images of the tool in its first version. Complexity, PMD, and Duplication displayed changes on a per day base. Which is why on some days the bar is negative as this indicated students had reduced complexity, or removed PMD messages.

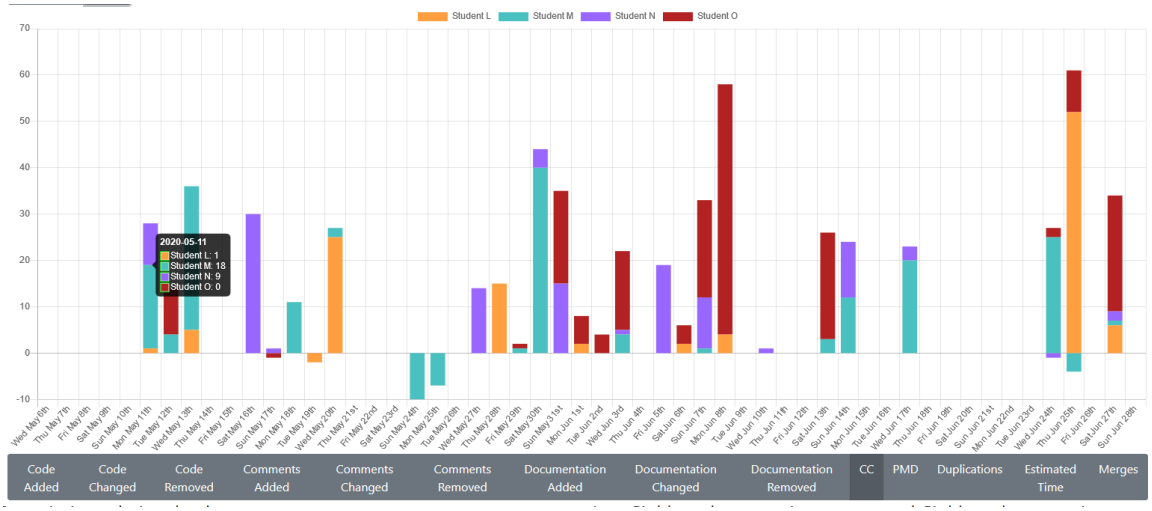

Figure 32: First version, Complexity

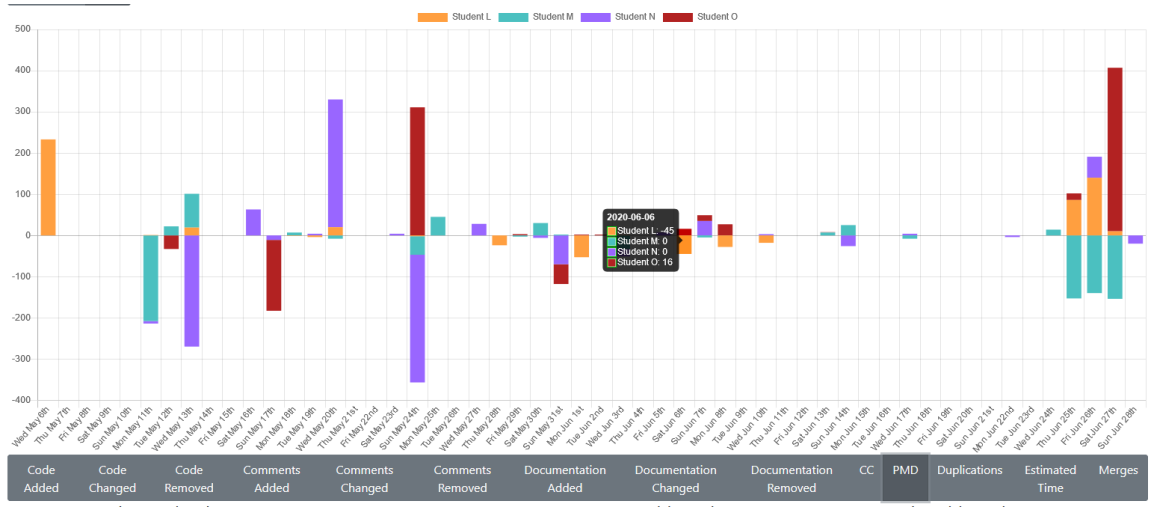

Figure 33: First version, PMD

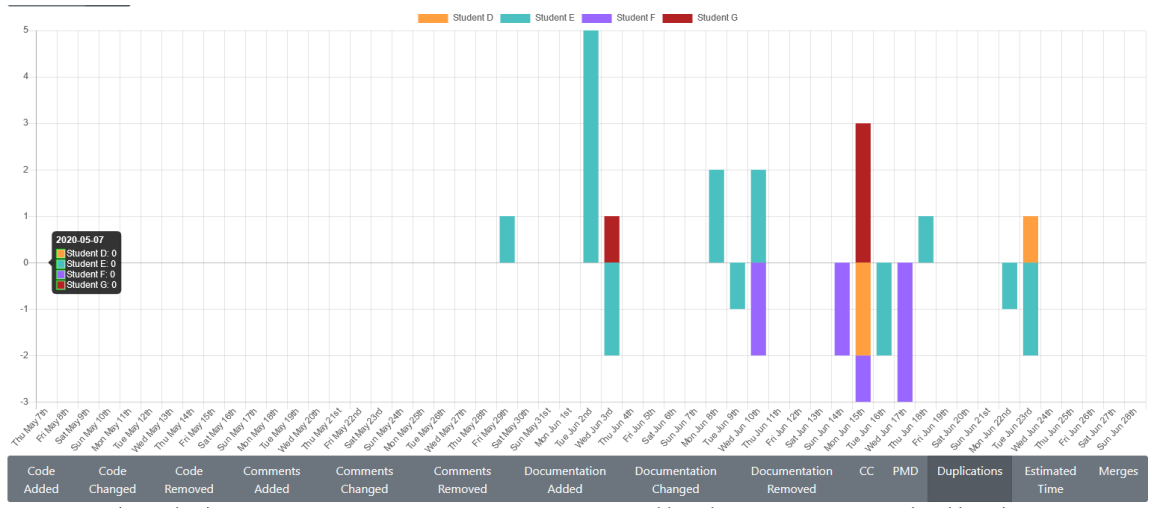

Figure 34: First version, Code Duplication from a different project with more data

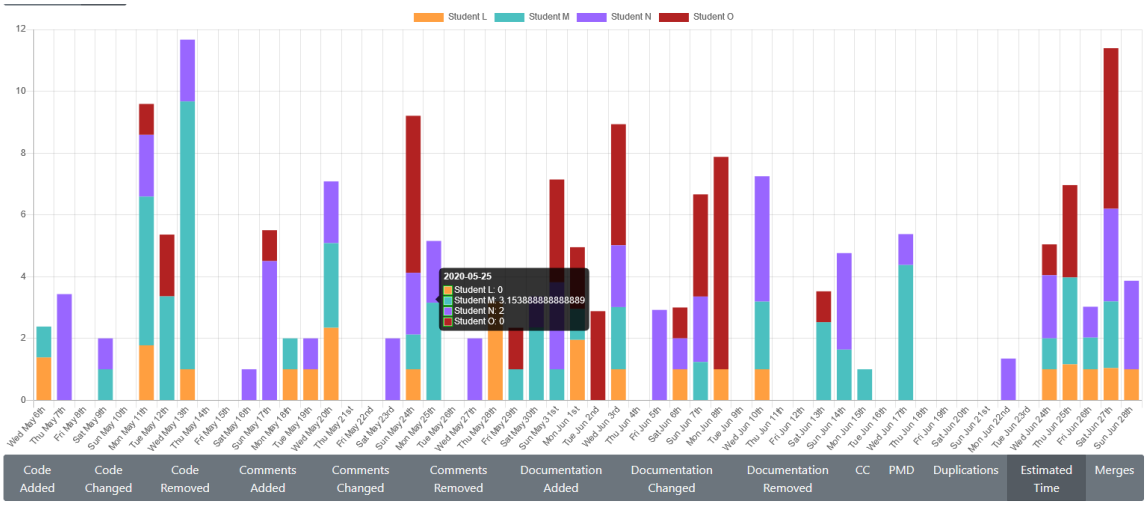

Figure 35: First version, Estimated Hours

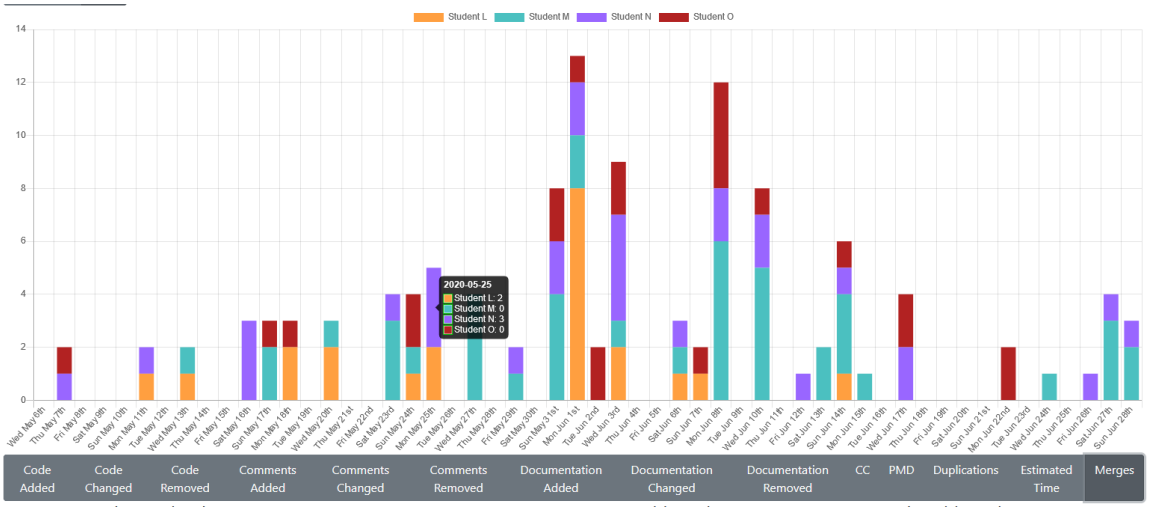

Figure 36: First version, Merges

B. Interview Protocol

Basic Structure

- Intro - 5 min
  - Introduction of research
  - It is about how the tool affected their guidance
  - We will discuss how it was utilized and whether or not they noticed any changes
  - I will be recording the interview (using teams) and also making notes.
  - Make sure I have consent form.
  - Ask if they have any questions before we start.
- Guidance - 10 min
  - Establish a baseline of how they normally do their guidance.
  - See questionlist for potential questions.
- Usage - 10 min
  - How did the tool change the way they did guidance
  - See questionlist for potential questions.
- Metrics - 10 min
  - Which metrics did they find the most valuable.
  - See questionlist for potential questions.
- Evaluation - 15 min
  - Overal evaluation of the tool and the metrics.
  - See questionlist for potential questions.
- Finalize - 5 min
  - Closing
  - What would you like to add before I stop recording?

Potential Questions

Guidance

- How long have you been doing project guidance?
- What sort of project guidance have you done in the past?
- How did you approach guiding those projects?
- How did you establish individual contribution in those project?
- How did you do individual guidance in those projects?

- How did you deal with under-/over-performing individuals?

Usage

- When did you use the metrics during guidance?
- How did you implement the metrics during guidance?
- Did you notice a difference in approach to guidance due the tool?
  - If yes, in what way?
  - If no, why not?
- How do did students respond to the metrics?
- Was the behavior of students different from previous experiences, and if so in what way?

Metrics

- Which metrics did you rely on the most during guidance?
- Which metrics would you have liked to also have available?
- Would you have liked to have seen metrics presented in a different way?
- Depending on time step through each metric.
- What did you think of the alternate version introduced for the last sprint?

Evaluation

- How was your overall experience using the metrics?
- Would you want to use this method in future/other projects?
- What do you feel would improve usage of these metrics?
- What would you like to see in the tool usage?

C. Survey Design

This quartile you worked on a project with your peers to develop a game in Android. During this project we started using a tool for the first time and would like your opinion on it.

You have seen the reports from your teacher containing metrics mined from the git repository that you worked on. If you have not seen these reports please contact Jan Jaap Sandee (JSA06) with your group name so you can gain access to see them (or ask your teacher during the class), before continuing this survey.

The following survery will be used to see how you experienced the usage and what improvements we can make. It will take a maximum of 5 minutes to fill out.

The data will be used for a master research on group guidance by Jan Jaap Sandee. The information will also be used to make improvements where needed.

The data is collected anonymously please fill it out based on your personal experience.

- **The metrics generated by the tool were discussed by the teacher during guidance sessions:**

  Options: Never / Sometimes / About half the time / Most of the time / Every Time

- **The group discussed the metrics generated by the tool on their own:** *Options:* Never / Sometimes / About half the time / Most of the time / Every Time

- **The metrics generated by the tool were used to decide on task division within the group:**

  Options: Never / Sometimes / About half the time / Most of the time / Every Time

- **For each metric state whether or not they were clear to you in what they represented:**

  Options: Not Clear / Kind of Clear / Fairly Clear / Very Clear

  Metrics: Lines of code, Comments, Documentation, Time of Commit, Complexity, Method Size, Git Blame / Code Division

- **For each metric state whether or not they reflected your experience during the project:**

  Options: Not accurate / Kind of accurate / Fairly accurate / Very accurate

  Metrics: Lines of code, Comments, Documentation, Time of Commit, Complexity, Method Size, Git Blame / Code Division

- *Pair programming is the act of working on code together with one person typing (and committing) and the other person observing and thinking out loud.*

  Which of the following reflects your experience?

  Options:

  - I did not participate in pair programming
  - Mostly typing and committing
  - Evenly divided between typing and observing
  - Mostly observing and thinking out loud

- **How did the metrics generated by the tool affect the way you worked on this project?**

  Open text question

- **How should we use this tool in future projects? Should we add or change something in the tool or the way we use it?**

  Open text question